Interopérabilité des environnements virtuels 3D : modèle de réconciliation des contenus et des composants logiciels

Rozenn Bouville

Thèse

THÈSE INSA Rennes
sous le sceau de l'Université Européenne de Bretagne
pour obtenir le titre de
DOCTEUR DE L'INSA DE RENNES
Spécialité : Informatique

présentée par
Rozenn BOUVILLE
ÉCOLE DOCTORALE : MATISSE
LABORATOIRE : IRISA – UMR6074
N° d'ordre : D12–37

Interopérabilité des environnements virtuels 3D : modèle de réconciliation des contenus et des composants logiciels

Thèse soutenue le jeudi 20 décembre 2012
devant le jury composé de :

Kadi BOUATOUCH
Professeur à l'Université de Rennes 1 / *Président du jury*
Florent DUPONT
Professeur à l'Université Claude Bernard de Lyon / *Rapporteur*
Jean-Pierre JESSEL
Professeur à l'Université Paul Sabatier de Toulouse / *Rapporteur*
Marius PREDA
Maitre de Conférences à l'Institut Télécom Sud Paris / *Examinateur*
Bruno RAFFIN
Chargé de Recherche INRIA dans l'équipe MOAIS à Grenoble / *Examinateur*
Jérôme ROYAN
Ingénieur de Recherche à Orange Labs Rennes / *Co-encadrant de thèse*
Thierry DUVAL
Maitre de Conférences à l'Université de Rennes 1 / *Co-encadrant de thèse*
Bruno ARNALDI
Professeur à l'INSA de Rennes / *Directeur de thèse*

"Reserving judgements is a matter of infinite hope."

F. SCOTT FITZGERALD

REMERCIEMENTS

AU terme de ces trois riches années, il est délicat de remercier chacun à la mesure de sa contribution à mon travail de thèse, que cette contribution soit d'ordre technique, didactique, psychique, matérielle ou encore administrative (il y a beaucoup de papiers à remplir durant une thèse...). Je m'en tiendrais donc à l'ordre traditionnel en tâchant simplement de n'omettre personne.

Je remercie Kadi BOUATOUCH, Professeur à l'Université de Rennes 1, qui m'a fait l'honneur et la joie de présider ce jury. Je remercie Florent DUPONT, Professeur à l'Université Claude Bernard de Lyon, et Jean-Pierre JESSEL, Professeur à l'Université Paul Sabatier de Toulouse, d'avoir bien voulu accepter la charge de rapporteur. Je remercie Marius PREDA, Maitre de Conférences à l'Institut Télécom Sud Paris, et Bruno RAFFIN, Chargé de Recherche INRIA dans l'équipe Moais à Grenoble, d'avoir bien voulu juger ce travail.

Je remercie Bruno ARNALDI d'avoir dirigé ma thèse et mes deux encadrants, Thierry DUVAL et Jérôme ROYAN. Merci à toi, Jérôme, de m'avoir fait confiance pour mener à bien ces travaux de recherche malgré ma lourde filiation ;-). Merci à toi, Thierry, d'avoir accepté de m'encadrer alors que ma thèse avait démarré depuis un an. Enfin, merci à vous trois pour vos conseils et vos propositions éclairées ainsi que pour votre soutien. J'espère de tout cœur que vous avez apprécié autant que moi notre collaboration.

Je tiens ensuite à remercier mes premiers compagnons-thésards qui m'ont lâchement abandonné à la fin de ma première année pour cause de thèses soutenues : Noémie, Nicolas et Sébastien. Merci d'avoir partagé vos conseils et votre expérience de ces trois années si particulières. J'ai essayé de transmettre à mon tour ce savoir aux nouveaux doctorants que j'ai eu le plaisir de côtoyer : Pierre M., Pierre G. et Huyen. Je remercie également Cédric et Huyen de m'avoir fait une place dans leur bureau et de m'y avoir chaleureusement accueilli 1 an plus tard... Je souhaite aussi remercier de manière générale mais avec beaucoup de reconnaissance l'équipe ASAP/IACA d'Orange Labs et l'équipe VR4i au sein desquelles j'ai travaillé durant ces 3 ans.

Un immense merci à mon cher et tendre mari, Julien, pour son soutien et son réconfort sans limite pendant ces trois années et pour longtemps encore j'espère. Merci à toi de m'avoir soutenue dans ce projet et d'avoir toujours été présent malgré tes journées bien remplies. Je remercie également mon fils, Yoni, même si sa contribution à ce travail reste très mineure voire plutôt perturbatrice ! Ta naissance a bouleversé nos vies et ma thèse mais pour mieux les mettre en perspective. Merci de nous permettre d'apprendre autant de choses chaque jour à ton contact.

Enfin je remercie ma famille et ma belle-famille qui ont elles-aussi, par leur soutien, largement contribué à l'accomplissement de mon travail. Un incommensurable merci à mes parents pour

avoir toujours cru en moi beaucoup plus que moi-même et à mon frère pour m'avoir ouvert la voix.

Pour conclure ces remerciements, je n'oublie pas mes amis qui ont toujours eu la patience d'écouter jusqu'au bout lorsqu'ils me demandaient le titre de ma thèse. Je remercie également ceux qui m'ont interrompu avant la fin. Il m'est impossible de les citer tous mais j'espère que chacun d'eux sait à quel point leur amitié m'est précieuse. J'ai une pensée toute particulière pour mon amie Alice qui m'offre son amitié depuis maintenant 24 ans. Que de chemin parcouru depuis l'époque où nous jouions avec nos poneys multicolores en plastique ...

TABLE DES MATIÈRES

INTRODUCTION

CETTE thèse se propose d'analyser et de contribuer à résoudre le problème de l'interopérabilité pour les environnements virtuels 3D. Tout d'abord nous exposerons de manière détaillée le contexte lié à notre problématique et les problèmes associés. Dans le paragraphe suivant, nous définirons l'interopérabilité et discuterons des enjeux liés à sa mise en place pour les environnements virtuels 3D. Enfin nous préciserons notre cadre d'étude et terminerons cette introduction par une description succincte des chapitres suivants.

CONTEXTE ET PROBLÉMATIQUE

Les environnements virtuels 3D sont des applications de réalité virtuelle qui ont, au fil des années, gagné en nombre et en diversité d'usages. Cette technologie a en effet trouvé des applications dans divers domaines qui vont du divertissement à des usages professionnels de haute précision. Leur diversité et leur nombre posent un problème pour lequel il n'existe pas de solution universelle : l'impossibilité de communiquer d'un environnement virtuel à un autre. Il est en effet extrêmement difficile d'échanger des données entre deux environnements virtuels mais également d'accéder à plusieurs environnements virtuels par l'intermédiaire d'une même application cliente.

Cet état de fait pose de nombreux problèmes qui freinent la diffusion et l'adoption des environnements virtuels 3D :

- **la réutilisation de données** : l'impossibilité d'échanger et de partager directement des données d'un environnement virtuel à un autre empêche la réutilisation des données et plus précisément des contenus des environnements virtuels. En effet, bien qu'il existe une large offre d'outils permettant de produire des contenus 3D, ceux-ci sont plus ou moins intuitifs à manipuler. Il est donc très intéressant de pouvoir réutiliser des modèles existants car leur réalisation est une tâche coûteuse et consommatrice de temps ;
- **la réutilisation et l'interchangeabilité des composants** : de même qu'il est pratiquement impossible de réutiliser directement des contenus 3D, il en va de même pour les composants logiciels qui permettent de générer l'environnement virtuel. Ceux-ci sont en effet compatibles avec un petit sous-ensemble d'autres composants et de contenus 3D ce qui limite leur réutilisation dans un autre environnement virtuel. Il est par conséquent souvent nécessaire de développer de nouveaux composants pour chaque environnement virtuel ce qui majore encore leur coût et leur temps de production ;
- **le recours à la conversion** : une solution couramment utilisée pour réutiliser les contenus et les composants consiste à convertir les données utilisées mais la conversion est souvent synonyme d'erreurs et de pertes de données qui peuvent rendre les contenus convertis inexploitables ;

- **les problèmes de portage** : le manque d'interopérabilité pose aussi problème lorsqu'un environnement virtuel doit être porté sur un autre système (nouveau client ou nouveau terminal) ;
- **les difficultés d'accessibilité** : les environnement virtuel 3D sont généralement accessibles au travers d'une application cliente qui leur est généralement propre. Pour accéder à un autre environnement virtuel, il est nécessaire d'utiliser une autre application cliente.

L'interopérabilité permet de résoudre tous ces problèmes de façon à accroître et à simplifier l'utilisation des environnements virtuels 3D. Il faut cependant distinguer l'interopérabilité entre les applications des environnements virtuels 3D et l'interopérabilité entres les composants des environnements virtuels 3D. Nous nous intéressons ici au second cas, nous cherchons à proposer une solution pour l'interopérabilité entre les contenus et les composants logiciels des environnements virtuels 3D mais aussi entre les contenus eux-mêmes et entre les composants logiciels eux-mêmes.

QU'EST CE QUE L'INTEROPÉRABILITÉ ?

DÉFINITION

Le terme "*interoperability*" est apparu en 1977 et vient de la contraction de deux mots ; "*inter*" mot latin signifiant "entre" et "*operability*" qui pourrait être traduit par "utilisable". Le terme anglais "interoperability" a été traduit par interopérabilité mais une traduction plus proche de sa signification originelle serait *interutilisabilité*. C'est une notion que l'on retrouve aujourd'hui dans de nombreuses branches de l'informatique. On dit que deux systèmes sont interopérables lorsque ceux-ci peuvent fonctionner et communiquer ensemble sans qu'il soit nécessaire de modifier l'une ou l'autre des parties au préalable. Il faut faire la distinction entre l'*interopérabilité* et la *compatibilité*. L'interopérabilité entre deux systèmes, à l'inverse de la compatibilité, nécessite que ces deux systèmes puissent partager des données ou des ressources sans restriction d'accès. Deux systèmes peuvent être compatibles sans pour autant donner un accès direct à leurs données ou à leurs ressources comme c'est le cas lorsque deux systèmes propriétaires collaborent. Ils peuvent être utilisés conjointement mais une interface bloque l'accès direct à leurs données respectives.

LES ENJEUX DE L'INTEROPÉRABILITÉ

L'interopérabilité dans le monde de l'informatique est une notion cruciale. Elle est souvent déterminante pour l'adoption d'une nouvelle technologie et est également un gage de pérennité dans un domaine qui évolue extrêmement vite. Le succès du modèle du Web en est un bon exemple car il est en grande partie lié à son interopérabilité. En effet, la capacité du Web à pouvoir être accessible avec n'importe quel navigateur, sur n'importe quelle machine a permis une adoption massive et rapide de cette technologie.

La notion d'interopérabilité est souvent liée à la mise en place de normes qui vont contraindre les développements futurs. Les règles définies dans la norme vont permettre d'assurer l'interopérabilité entre deux entités développées dans des contextes différents. L'adoption dès sa création du format HTML et du protocole HTTP sont les normes qui ont structuré le développement du Web. De ce fait, il est parfois reproché à l'interopérabilité de limiter les innovations techniques car les contraintes imposées bride les nouveaux développements. Cependant, à l'inverse, le manque d'interopérabilité peut mener à une situation de monopole, elle aussi néfaste à l'innovation. En effet si un produit non interopérable est adopté massivement cela peut bloquer les innovations concurrentes. L'interopérabilité est ainsi une situation souhaitable dans la majorité des cas mais celle-ci ne doit pas se faire au prix d'un blocage des évolutions potentielles d'un système.

Si l'on rapporte cette notion aux environnements virtuels, elle se traduit par la possibilité de permettre à une entité d'interagir avec un environnement virtuel différent de celui pour lequel elle a été conçue. L'interopérabilité des environnements virtuels 3D permettrait non seulement

de réutiliser directement les contenus et les composants des environnements virtuels existants sans les convertir mais également de capitaliser les fonctionnalités et les avantages de chacun pour créer des environnements virtuels plus riches et plus performants. Ceci améliorerait significativement la production des environnements virtuels mais faciliterait également leur utilisation. L'interopérabilité des environnements virtuels 3D rendrait en effet possible la navigation et l'interaction avec une multitude d'environnements virtuels en utilisant un client de visualisation unique.

CADRE D'ÉTUDE

Dans le contexte des environnements virtuels 3D, une solution d'interopérabilité pour les contenus et les composants logiciels des environnements virtuels 3D nous semble devoir respecter cinq critères essentiels :

1. *une utilisation directe des contenus* : c'est-à-dire une utilisation sans conversion de la plupart des contenus ;
2. *une utilisation directe des composants logiciels* : c'est-à-dire une utilisation sans modification de la plupart des composants logiciels utilisés par une plateforme d'environnement virtuel 3D ;
3. *une communication entre contenus et composants* : la communication entre l'ensemble des contenus et l'ensemble des composants de rendu d'un environnement virtuel 3D ;
4. *une communication inter-contenus* : la communication entre l'ensemble des contenus d'un environnement virtuel 3D ;
5. *une communication inter-composants* : la communication entre l'ensemble des composants logiciels d'un environnement virtuel 3D.

Ces cinq critères constituent le cadre initial de nos travaux. Nous nous sommes donc basé sur ces critères pour concevoir notre solution puis l'évaluer.

DESCRIPTION PAR CHAPITRE

Ce manuscrit est découpé en deux parties : une partie dédiée à l'état de l'art et une partie dédiée à la contribution. L'état de l'art regroupe trois chapitres, le chapitre 1 donne un aperçu de la diversité des environnements virtuels au travers d'une classification. Le chapitre 2 recense et commente les solutions existantes. Enfin, le chapitre 3 analyse les contenus 3D et les composants logiciels des environnements virtuels 3D. Notre contribution regroupe quant à elle quatre chapitres. Le chapitre 4 expose le principe fondateur de notre solution puis le chapitre 5 explique de façon détaillée le système sur laquelle se base notre solution que nous avons nommée le SGA pour Adaptateur de Graphes de Scène (ou *Scene Graph Adapter* en anglais). Ensuite, le chapitre 6 présente l'implémentation que nous avons réalisée dans le but d'évaluer notre solution. Enfin, le chapitre 7 propose un modèle qui s'intègre à notre système afin de permettre la communication entre les contenus 3D.

Première partie

État de l’art

Chapitre 1

LES ENVIRONNEMENTS VIRTUELS 3D

DE la Sensorama Machine de Morton L. Heilig [Hei62] en 1956 aux CAVE™[CNSD+92, CNSD93] actuels en passant par les environnements virtuels textuels tels que MUD [Bar78] ou par les métavers tels que Second Life, les environnements virtuels ont évolué au gré des innovations techniques et trouvé au fil des années de multiples applications. Ils sont aussi divers par leurs natures, leurs dénominations et leurs domaines d'application. Ils sont aujourd'hui utilisés quotidiennement par de nombreuses personnes aussi bien pour des applications professionnelles, artistiques ou de divertissement. Dans ce chapitre, nous allons parcourir les différents termes utilisés pour les désigner. Ensuite, nous tenterons de donner un aperçu des différents types d'environnement virtuel 3D existants actuellement et de leurs domaines d'application au travers d'une classification des environnements virtuels 3D qui évalue pour chaque catégorie l'intérêt d'une solution d'interopérabilité.

1.1 PRÉSENTATION DES ENVIRONNEMENTS VIRTUELS 3D

La terminologie utilisée pour la notion d'environnement virtuel est étendue dans la littérature. On parle couramment d'environnement virtuel, de monde virtuel, d'espace virtuel ou d'univers virtuel. Tous ces termes englobent des notions similaires mais cependant nuancées qui ont trait à la réalité virtuelle. La définition de la réalité virtuelle qui fait référence et sur laquelle nous nous basons dans ces travaux est celle proposée dans "Le Traité de la Réalité Virtuelle" [FMB+06] :

Définition 1.1 *La **réalité virtuelle** est un domaine scientifique et technique exploitant l'informatique et des interfaces comportementales en vue de simuler dans un monde virtuel le comportement d'entités 3D, qui sont en interaction en temps réel entre elles et avec un ou des utilisateurs en immersion pseudo-naturelle par l'intermédiaire de canaux sensori-moteurs.*

Les termes de monde virtuel, espace virtuel, univers virtuel et environnement virtuel sont similaires et désignent le monde simulé dans une application de réalité virtuelle. On peut néanmoins constater quelques nuances en fonction du contexte d'utilisation et notamment de la communauté visée par ce contexte. Certaines communautés préféreront utiliser le terme d'environnement virtuel alors que d'autres parleront systématiquement de monde virtuel. En effet l'un ou l'autre de ces termes peut être préféré selon l'étendue et le réalisme du monde simulé : deux propriétés qui permettent de caractériser le monde. Dans le contexte de cette thèse, nous avons choisi d'utiliser le terme d'environnement virtuel 3D car sa signification est suffisamment vaste pour englober l'ensemble des applications 3D auxquelles les travaux présentés ici s'appliquent.

Selon les fonctionnalités demandées par la finalité de l'application de réalité virtuelle, un environnement virtuel 3D peut posséder les caractéristiques suivantes :

- l'utilisateur peut y avoir une représentation sous forme d'avatar ;

- l'utilisateur peut y créer de nouveaux objets ;
- l'utilisateur peut interagir avec d'autres utilisateurs ;
- l'environnement peut simuler d'autres sens humains (l'ouïe et le toucher notamment) ;
- l'environnement peut être partagé ;
- l'environnement peut être persistant.

D'autre part, un environnement virtuel 3D est plus ou moins complexe et nécessite la mise en œuvre de nombreuses techniques qui vont permettre d'optimiser le compromis entre résultat souhaité et capacité du matériel informatique ciblé. En effet, ce dernier étant plus ou moins puissant, il ne permet pas toujours de réaliser la totalité des spécifications demandées lors de la conception.

1.2 CLASSIFICATION DES ENVIRONNEMENTS VIRTUELS 3D

Dans cette partie, nous avons répertorié les principaux types d'environnement virtuel 3D. Nous discuterons pour chacun d'eux si une solution d'interopérabilité est pertinente et souhaitée. En effet, l'ensemble des domaines d'application des environnements virtuels 3D est tellement vaste et hétérogène qu'il regroupe des utilisateurs aux intérêts parfois divergents.

1.2.1 LES JEUX VIDÉOS

Le divertissement est historiquement l'un des premiers usages des environnements virtuels. Il est donc logique que le domaine des jeux vidéos soit le principal créateur d'environnement virtuel 3D de nos jours. Les différents jeux vidéo proposent plus ou moins de fonctionnalités selon le type de jeu pour lequel ils ont été créés. Les principaux types de jeux vidéo sont :

- **les jeux de tirs subjectifs ou FPS (First-Person Shooters)** : le point de vue est celui de l'utilisateur qui est, en général, un personnage muni d'une arme. (Exemple : *Quake*, *Unreal Tournament*, *Half-Life*, ...). Dans ces jeux, le monde est visuellement très réaliste et relativement étendu.
- **les jeux de plateforme** : ce sont les jeux dont l'action est basée sur un personnage tierce (Exemple : *Space panic*, *Donkey Kong*, *Super-Mario Brothers*, ...). Pour ce type de jeu, le monde n'est pas toujours tridimensionnel, son réalisme importe peu et il est plutôt restreint.
- **les jeux de combat** : ces jeux sont généralement jouables à deux et les deux personnages doivent s'affronter au cours d'un combat (Exemple : *Soul Calibur*, *Tekken*, *Mortal kombat*, ...). Le monde y est restreint et le réalisme est de moindre importance.
- **les jeux de course** : ces jeux se déroulent sur un circuit et le joueur doit concourir à une courses de voiture ou tout autre type de véhicule (Exemple : *Grand Turismo*, *Need For Speed*,...). Pour ce type de jeux, la superficie du monde se limite aux circuits mais le réalisme visuel et physique y est généralement essentiel.
- **les jeux de stratégies ou RTS (Real Time Strategy)** : dans ce type de jeu, le joueur doit conquérir un monde en commandant son unité de combat (Exemple : *Age Of Empires*, *Warcraft*, *Command Conquer*, ...). Ce genre de jeu attache peu d'importance au réalisme du monde mais celui-ci peut être très étendu.
- **les jeux massivement multijoueurs ou MMOG (Massively Multiplayer Online Games)** : ce sont des jeux en ligne où des utilisateurs du monde entier s'affrontent dans un monde virtuel (Exemple : *World Of Warcraft*, *EverQuest*, *Star Wars Galaxies*, ...). Ce dernier est généralement très vaste et continue de s'étendre à mesure que des mises à jour sont proposées, cependant son réalisme n'est pas primordial.
- **les jeux de simulation de vie** : ce type de jeu n'a pas de but précis, il s'agit de gérer la vie d'un personnage ou d'une famille. (Exemple : *Les Sims*, *SimCity*). Le monde est restreint et son réalisme est important.
- **les jeux de rôle ou RPG (Role-Playing Games)** : dans ces jeux, le joueur incarne un ou plusieurs personnages et doit mener à bien une quête. Ils peuvent être massivement multi-

joueurs auquel cas on peut les désigner par le terme MMORPG (Massively Multiplayer Online Role-Playing Games) (Exemple : *Dragon Quest*). Le monde dans lequel le joueur évolue est souvent très vaste et ce d'autant plus s'il est multijoueur mais on attache peu d'importance à son réalisme.

- **les jeux de sport** : jeux de simulation du sport (Exemple : *FIFA, NBA*) qui requièrent un environnement réaliste mais peu étendu.

(a) Quake

(b) Super Mario

(c) Tekken

(d) Gran Turismo

(e) Age Of Empire

(f) World Of Warcraft

(g) Les Sims

(h) Dragon Quest

(i) FIFA

FIGURE *1.1 – Les différents types de jeu : un jeu de tir subjectif (1.1a), un jeu de plateforme (1.1b), un jeu de combat (1.1c), un jeu de course (1.1d), un jeu de stratégie (1.1e), un jeu massivement multijoueur (1.1f), un jeu de simulation de vie (1.1g), un jeu de rôle (1.1h) et un jeu de sport (1.1i).*

Le tableau 1.1 indique pour chaque type de jeu ses critères en terme de réalisme et d'étendue du monde ainsi que les autres caractéristiques qu'il peut présenter.

	TYPE DE JEU								
Propriété	Tir subjectif	Plateforme	Combat	Course	Stratégie	MMOG	Simulation de vie	Rôle	Sport
ÉTENDUE	**	**	*	*	***	***	**	***	*
RÉALISME	***	*	**	***	*	*	**	*	***
AVATAR	oui	oui	oui	oui	oui	oui	oui	oui	oui
POSSIBILITÉ DE CRÉER DE NOUVEAUX OBJETS	non	non	non	non	oui	parfois	oui	parfois	non
INTERACTIONS AVEC AUTRES UTILISATEURS	non	parfois	oui	oui	parfois	oui	parfois	parfois	oui
AUTRE(S) SENS SIMULÉ(S)	ouïe	ouïe	ouïe	ouïe	ouïe	ouïe	ouïe	ouïe	ouïe
MONDE PARTAGÉ	non	non	non	non	parfois	parfois	parfois	parfois	non
MONDE PERSISTENT	non	non	non	non	parfois	parfois	non	parfois	non

TABLE *1.1 – Comparaison des propriétés des principaux types de jeu*

On peut constater à la lecture du tableau qu'au sein d'une même catégorie d'environnement virtuel les propriétés et les fonctionnalités exigées sont très diverses. Ceci rend une solution d'interopérabilité complexe à mettre en place car celle-ci devra prendre en compte cette hétérogénéité. Cependant, ce n'est pas le seul obstacle à sa mise en œuvre. En effet, l'industrie du jeu vidéo est un secteur économique prospère, contrôlé par un cercle relativement restreint de studios. Ceux-ci ne souhaitent pas voir une interopérabilité possible entre leurs produits car cela remettrait en cause leur modèle économique.

1.2.2 LES MÉTAVERS

Les métavers sont des mondes virtuels 3D dans lesquels les utilisateurs évoluent sous forme d'avatars personnalisables. Dans ces environnements, l'immersion se fait entièrement via l'avatar de l'utilisateur. Le terme *métavers* provient du roman de science-fiction "Le samouraï virtuel" [1] de Neal Stephenson [Ste00] écrit en 1992. Ils divergent des MMOG car ils sont continus et persistants [**?**]. Si certains MMOG le sont également, c'est à une échelle et une complexité moindre. En effet, ils sont moins étendus que les métavers. De plus, le contenu des métavers est créé par les utilisateurs eux-mêmes. Ceci implique un nombre considérable de contenus sans cesse en évolution au gré des créations et des modifications des utilisateurs.
La fonction principale d'un métavers est de mettre en relation des utilisateurs réels du monde entier par l'intermédiaire de leurs avatars afin qu'ils y développent des activités éventuellement lucratives. Les utilisateurs forment une communauté au sein de laquelle ils nouent les même relations que dans le monde réel. Les métavers les plus importants sont devenus des micro-sociétés, ainsi, les plus développées ont mis en place un commerce virtuel ou v-commerce associé à une monnaie virtuelle.

Le premier métavers, *CitySpace* [2], fût lancé en 1993 lors du Siggraph. Depuis, nombre d'entre eux ont été mis en ligne dont *Le Deuxième Monde* [3] en 1994, *ActiveWorld* [4] en 1995 ou encore *There* [5] en 2004. Ils étaient au nombre de 500 en 2011 et on devrait en dénombrer plus de 800 à la fin de l'année 2012 d'après l'institut KZero [6]. D'après ce même organisme, les revenus engendrés par ce secteur économique sont à l'image de leur croissance en nombre ; d'un total de 4 milliards de $ en 2011, ils devraient atteindre 9 milliards de $ en 2013.
Le plus populaire à ce jour est *Second Life* [7] qui a été lancé en 2003. Celui-ci totalisait 2 millions de résidents en décembre 2006 [HLT08] et en comptait 15 millions en 2009 [BSPM09]. Il semblerait

1. Titre original "Snow Crash"
2. http://en.wikipedia.org/wiki/Cityspace
3. http://www.web3d-fr.com/articles/Portraits/2002-LEDIBERDER/2m.php
4. http://www.activeworlds.com/
5. http://www.there.com/
6. http://www.kzero.co.uk
7. http://secondlife.com/?v=1.1

cependant que son succès tende à s'estomper, après une période très faste en 2005 et 2006, faisant dire à certains que "second life is dead" [Liv11]. Cependant, ce monde continue à attirer de nouveaux utilisateurs (en moyenne 24 000 nouveaux utilisateurs par jour à la fin de l'année 2011) et reste rentable pour son éditeur avec 100 millions de dollars de revenus pour 2011.

Propriété	
Étendue	***
Réalisme	*
Avatar	oui
Possibilité de créer de nouveaux objets	oui
Interactions avec autres utilisateurs	oui
Autre(s) sens simulé(s)	ouïe
Monde partagé	oui
Monde persistent	oui

TABLE *1.2 – Propriétés des métavers*

Pour les éditeurs des métavers, une solution d'interopérabilité ne présente d'intérêt que lorsque le modèle économique sur lequel il repose le permet. Cependant, une solution d'interopérabilité permettrait aux utilisateurs de partager leurs contenus et de préserver leur identité entre différents métavers.

1.2.3 L'APPRENTISSAGE EN LIGNE (E-LEARNING) ET LES JEUX SÉRIEUX (SERIOUS GAMES)

Les environnements virtuels ne sont pas utilisés uniquement dans le domaine du divertissement, ils servent de plus en plus à des fins d'apprentissage par la simulation. C'est ce qui est désigné depuis peu sous le terme d'"e-learning". Cependant, l'idée d'utiliser un environnement virtuel dans le domaine de l'éducation n'est pas nouvelle. Dès 1973, les jeux éducatifs *The Oregon Trail*[8] et *Lemonade Stand*[9] ont été développés dans ce but. Le premier proposait aux jeunes américains et canadiens de suivre la vie de pionniers en route vers l'Oregon en incarnant un chef de convoi. Le second initiait les élèves à la gestion d'un commerce en gérant un stand de limonade.

FIGURE *1.2 – The Oregon Trail*

Actuellement ce type d'application connaît un nouvel essor grâce à l'engouement pour les jeux sérieux (serious games). La définition d'un jeu sérieux varie selon les auteurs, certains considèrent que celui-ci implique obligatoirement la mise en place d'une règle du jeu ainsi que d'indications sur la façon d'y jouer à destination de l'utilisateur (ce que l'on résume souvent par l'utilisation de l'anglicisme "gameplay"). D'autres emploient ce terme pour désigner tout type de simulation à but d'apprentissage. Néanmoins les jeux sérieux intéressent de plus en plus de domaines depuis

8. http://en.wikipedia.org/wiki/The_Oregon_Trail
9. http://en.wikipedia.org/wiki/Lemonade_Stand

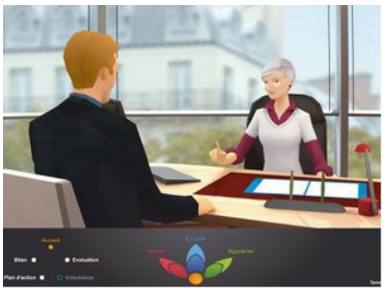
le succès de *America's Army* [10], un jeu de simulation d'entraînements militaires et de missions de combat, basé sur le moteur de jeu *Unreal Tournament* [11]. Il a été développé pour le compte de l'armée américaine afin d'inciter les joueurs à s'engager dans l'armée américaine. Ce jeu a été lancé en 2002 et totalisait 17 millions de téléchargements à l'issue de l'année 2004.

FIGURE *1.3 – America's Army*

Les environnements virtuels sont utilisés pour la formation dans des domaines tels que :

- **la sécurité** : sécurité civile et situations dangereuses (exemple : *Virtual Scylla*[SWGF09]) ;
- **la santé** : santé publique, formation thérapeutique (exemple : *Pulse ! !* [12]) ;
- **l'industrie** : production, maintenance et prévention des risques (exemple : *3D Réseaux* [13]) ;
- **le management** : entretiens RH, relation commerciale et gestion de projet (exemple : *Hair-Be12* [14] *Entre2* [15]) ;
- **la défense** : entraînement tactique, simulation et aide à la décision (exemple : *Marine Doom* [16]).

(a) Pulse ! !



(b) Entre2

FIGURE *1.4 – Exemples de jeux sérieux : Pulse ! !, un jeu de simulation pour la formation des médecins et Entre2, jeu d'entraînement aux entretiens professionnels.*

Une solution d'interopérabilité dans le domaine de l'apprentissage serait très profitable. Cela permettrait de réutiliser des contenus et de mettre en commun des éléments. La collaboration entre équipes serait ainsi facilitée.

1.2.4 LE WEB GÉOSPATIAL

Les environnements virtuels 3D sont également représentés dans ce que l'on peut appeler le web géospatial. Cette dénomination regroupe tous les services de cartographie 3D disponibles

10. www.americasarmy.com
11. http://www.unrealtournament.com/
12. http://www.breakawaygames.com/serious-games/solutions/healthcare/
13. http://www.b2b-games.com/3dreseaux/
14. http://www.hair-be12.com/accueil_v2.aspx
15. http://www.opcalia-idf.com/pages/entre2/index.php
16. http://en.wikipedia.org/wiki/Marine_Doom

Propriété	
Étendue	*
Réalisme	***
Avatar	parfois
Possibilité de créer de nouveaux objets	parfois
Interactions avec autres utilisateurs	parfois
Autre(s) sens simulé(s)	oui selon le domaine
Monde partagé	non
Monde persistent	non

TABLE *1.3 – Propriétés des environnements d'apprentissage*

sur le web permettant de visualiser des données géolocalisées issues des systèmes d'information géographique ou SIG. On peut classer dans cette catégorie Google Earth [17], le service de géolocalisation de Google ainsi que son concurent Microsoft, Bing Maps 3D [18] mais aussi l'initiative française de l'IGN, le Géoportail [19] .

(a) Bing Maps 3D (b) Google Earth

FIGURE *1.5 – Vues de New York dans Bing Maps 3D et Google Earth*

Ces services rencontrent un fort succès, Google célébrait l'année dernière le fait que Google Earth avait été téléchargé plus de 1 billion de fois.

Propriété	
Étendue	***
Réalisme	***
Avatar	non
Possibilité de créer de nouveaux objets	parfois
Interactions avec autres utilisateurs	non
Autre(s) sens simulé(s)	non
Monde partagé	parfois
Monde persistent	parfois

TABLE *1.4 – Propriétés des environnements de géovisualisation*

17. http://www.google.com/earth/index.html
18. http://www.bing.com/maps/
19. http://www.geoportail.fr/

Ces services proposent souvent à leurs utilisateurs d'ajouter de nouveaux contenus (des modèles 3D mais aussi des annotations géolocalisées) ce qui crée une communauté au sein de cet environnement virtuel. Ces services sont concurrents et il est probable qu'une solution d'interopérabilité n'aurait que peu d'intérêt pour leurs éditeurs alors que les utilisateurs sont très demandeurs. Néanmoins, il existe des solutions concurrentes telles que CityGML qui sont quant à elles interopérables car s'appuyant sur des standards ouverts et qui peuvent avoir un fort potentiel.

1.2.5 La Conception Assistée par Ordinateur

La CAO (Conception Assistée par Ordinateur) est un autre domaine où les environnements virtuels 3D sont également utilisés. Il s'agit de logiciels utilisés pour modéliser des objets avant leur fabrication. Le monde virtuel permet de simuler l'objet dans son environnement afin de le tester. La simulation est parfois très complexe, notamment dans le domaine de la mécanique. En effet, tout type de contrainte peut être exprimé et modélisé (fonctionnalités, matériaux, capacité d'assemblage, fabrication, ...). Cette application des mondes virtuels existe depuis les années 60 avec AutoCAD et on trouve aujourd'hui de nombreux logiciels très évolués comme *CATIA* [20] de Dassault Systèmes, *SOLIDWorks* [21] ou encore *Inventor* [22] de AutoDesk.

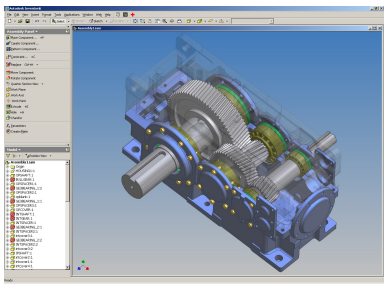


FIGURE *1.6 – Copie d'écran de AutoDesk Inventor*

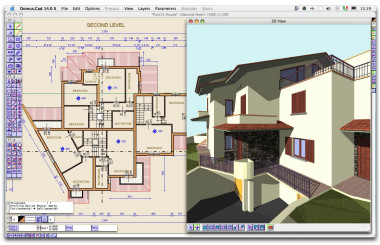
La CAO est utilisée dans bien d'autres domaines tels que l'électronique (Altium Designer [23]), l'architecture (AutoCAD [24], Domus [25]), les travaux publics, la modélisation moléculaire (Rasmol [26]), l'ameublement (Spazio3D [27]), la confection (Modaris [28]) mais aussi l'orthopédie (OrthoView [29]).

Pour ce type d'application, le manque d'interopérabilité est un problème crucial et récurrent. Il engendre des retards très coûteux en particulier lorsqu'un projet met en collaboration plusieurs partenaires utilisant des logiciels aux données incompatibles.

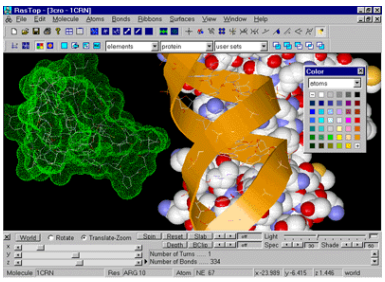
1.3 Synthèse et conclusion

Ce tour d'horizon des environnements virtuels 3D nous montre à quel point les environnements virtuels sont hétérogènes. Ils recouvrent des usages très différents aussi ils ne portent pas le même intérêt aux bénéfices qu'offre l'interopérabilité. Les utilisateurs d'environnements virtuels de CAO auraient tout intérêt à trouver une solution d'interopérabilité en particulier pour

20. http://www.3ds.com/products/catia/welcome/
21. http://www.solidworks.fr/default.htm
22. http://usa.autodesk.com/adsk/servlet/pc
23. http://www.altium-eda.fr/
24. http://usa.autodesk.com/adsk/servlet/pc
25. http://www.interstudio.net/DomusCadE.html
26. http://rasmol.org/
27. http://www.spazio3d.com/
28. http://www.lectra.com/en/fashion_apparel/products/modaris_fashion.html
29. http://www.orthoview.com/



(a) Modélisation d'architecture avec Domus



(b) Modélisation moléculaire avec Rasmol

FIGURE *1.7 – Exemples de logiciels de CAO.*

Propriété	
Étendue	*
Réalisme	***
Avatar	non
Possibilité de créer de nouveaux objets	oui
Interactions avec autres utilisateurs	parfois
Autre(s) sens simulé(s)	toucher
Monde partagé	non
Monde persistent	non

TABLE *1.5 – Propriétés des environnements de CAO*

des projets collaboratifs. Par contre, ce n'est pas le cas des concepteurs d'environnement virtuel pour des domaines très fermés tels que le jeu vidéo. Ceci explique pourquoi des solutions n'ont pas été proposées ou imposées plus tôt. Une solution d'interopérabilité doit être facilement acceptée. Pour cela, elle doit à la fois proposer un excellent compromis technique mais également représenter un intérêt pour les acteurs du secteur visé. De cette façon, elle pourrait être acceptée par le plus grand nombre possible d'utilisateurs ce qui la rendrait plus efficace et plus facilement évolutive. Dans le chapitre suivant, nous allons passer en revue les solutions d'interopérabilité existantes pour les contenus, les composants logiciels ainsi que les solutions plus globales. Nous les analyserons afin d'élaborer une solution conforme à notre cadre d'étude.

Chapitre 2

L'INTEROPÉRABILITÉ DES ENVIRONNEMENTS VIRTUELS 3D

Le chapitre 1 a présenté les environnements virtuels 3D et ce qui fait leur hétérogénéité. Cette hétérogénéité rend la mise en place d'une solution d'interopérabilité complexe car pour être facilement adoptée, elle doit prendre en compte cette hétérogénéité tout en restant relativement simple à mettre en œuvre. Dans ce contexte précis, l'objectif recherché est de s'affranchir des problèmes de compatibilité des contenus et des composants logiciels de rendu des environnements virtuels 3D afin de pouvoir les utiliser conjointement, les permuter et leur permettre d'interagir comme illustré dans la figure 2.2. Dans ce chapitre, nous allons examiner les solutions existant dans la littérature. Nous étudierons d'une part les solutions pour l'interopérabilité des contenus 3D puis les solutions pour l'interopérabilité des composants logiciels de rendu. Enfin nous présenterons les solutions qui proposent une approche globale pour les environnements virtuels. Pour chaque catégorie de solutions, nous les avons classées en trois types comme proposé dans l'article de Haslhofer et Klas [HK10] : les solutions qui préconisent de s'entendre sur un modèle, les solutions qui conseillent la mise en place d'un métamodèle et les solutions qui proposent de réconcilier les modèles. Ces solutions sont illustrées dans la table 2.1.

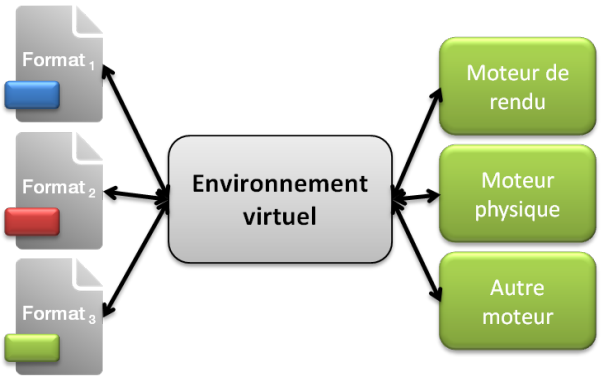
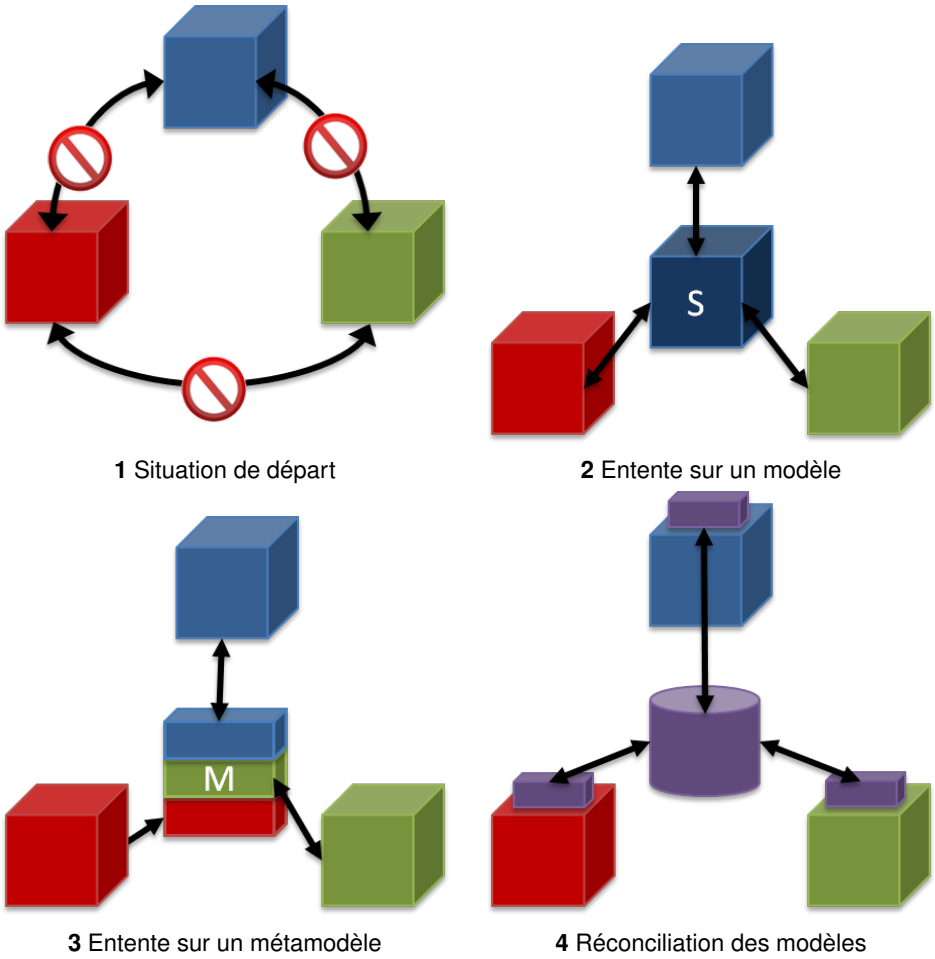


FIGURE 2.1 – *On souhaite pour un environnement virtuel donné pouvoir utiliser n'importe quel contenu 3D quel que soit son format ainsi que n'importe quel composant de rendu.*

L'*entente sur un modèle* consiste à établir un consensus en mettant en place un standard. C'est une méthode intuitive, efficace techniquement et bien reconnue économiquement. Cependant, afin d'être largement acceptée, elle nécessite le support d'une institution comme le W3C ou l'ISO par exemple. La mise en place d'un standard entraîne un processus de normalisation qui est souvent moins rapide que l'évolution des technologies, cela peut parfois nuire à son adoption. De



1 Situation de départ **2** Entente sur un modèle

3 Entente sur un métamodèle **4** Réconciliation des modèles

TABLE *2.1* – *Les différents types de solutions d'interopérabilité*

plus, ce type de solution est souvent un consensus minimum qui bloque son adoption auprès de certains acteurs qui ont leur propre solution propriétaire déjà mise en place. L'*entente sur un métamodèle* est une alternative qui tente de pallier les faiblesses liées au consensus établi lors de l'entente sur un modèle. Le métamodèle prend donc en compte tous les modèles existants et les intègre dans son propre modèle. C'est une solution efficace mais très complexe à mettre en œuvre et ce d'autant plus que le nombre de modèles est élevé. La dernière catégorie, la *réconciliation des modèles*, rend les modèles interopérables en essayant de réconcilier leurs hétérogénéités. C'est la méthode la plus complexe car elle nécessite une connaissance exhaustive de la nature de ces hétérogénéités. Elle demande donc une analyse préalable afin d'identifier une voie de réconciliation. Elle est souvent moins efficace que les autres types de solution mais elle n'en a pas les inconvénients car elle est à la fois globale et évolutive.

2.1 L'INTEROPÉRABILITÉ DES CONTENUS 3D

Nous nous intéressons dans cette partie aux solutions qui existent pour permettre l'interopérabilité des contenus 3D. Il s'agit de solutions qui tentent de proposer une alternative à la profusion des formats 3D.

2.1.1 ENTENTE SUR UN MODÈLE

Ce type de solution propose la mise en place d'un format standard. Dès les années 90, plusieurs projets ont cherché à mettre en place un format standard afin de répondre aux problèmes générés par la multiplicité des formats 3D. Le format OFF [Ros89], un format extensible ou encore le format P3D [WNA90], un format basé sur le langage Lisp, en sont deux exemples. Ces premières initiatives sont restées plutôt confidentielles et le premier réel effort de standardisation remonte à 1994 avec le format VRML (Virtual Reality Modeling Language) [CPC99] standardisé en 1996. Les objectifs visés par ce langage [LMM96] consistaient à créer un langage de réalité virtuelle ouvert et multi-plateforme pour le web. Cette première version a eu beaucoup de succès dans le monde industriel et est encore aujourd'hui utilisée bien qu'elle peine à être intégrée aux applications grand public. Les explications de ce succès mitigé sont multiples. Son manque d'extensibilité ainsi que la piètre qualité des exports des outils de modélisation seraient en partie responsables. On reprochait également à ce langage de générer des fichiers trop grands, de ne pas proposer de compression ni de streaming ceci entraînant une vitesse de téléchargement et d'affichage trop lente [FL05a]. Autre argument, la gestion de l'activité à travers des scripts compilés à la volée serait un peu lourde et aurait été un autre frein à l'adoption de VRML. Cependant, une raison majeure réside dans le fait que ce langage était très précurseur. En effet, il a été créé bien avant que les processeurs et le réseau disponible pour le grand public ne soient en mesure de gérer les scènes réalisables avec cette technologie [Pes94, OJ10]. De plus, l'argument selon lequel VRML serait un format volumineux est contredit par le succès actuel de Collada [AB06] qui génère des fichiers aussi grands, voire plus, que VRML. Quant à la mauvaise qualité des exports, la raison en est plus économique que technique. En effet, le secteur le plus prolifique concernant les mondes virtuels est le secteur du jeu vidéo. Les studios de jeux vidéo ne se sont pas intéressés à VRML car ce standard ouvert ne s'intégrait pas à leur modèle économique. Aussi les éditeurs de modeleurs 3D, dont la majorité de leur clientèle est constituée par les éditeurs de jeu vidéo, n'ont pas cherché à améliorer leurs exports. Le manque d'extensibilité a par ailleurs été comblé par la création de X3D [for06] en 2005, successeur de VRML, mais ceci n'a pas suffi à faire adopter massivement ce nouveau format 3D.

Un autre projet plus actuel, à l'initiative du consortium Khronos Group, WebGL [Khr11], propose une interface de programmation pour la programmation d'application 3D sur le Web. Il s'adresse à la communauté du web en permettant la déclaration et l'interprétation d'instructions OpenGL ES 2.0 directement dans le code javascript d'une page HTML. Pour cela, il utilise la balise "Canvas" de HTML5 et permet ainsi l'intégration directe de contenus 3D dans une page web. Cette technologie rencontre un fort succès et génère beaucoup d'attentes de la part de la communauté comme le montre son intégration à plusieurs navigateurs bien avant que les spécifications définitives ne soit publiées. Beaucoup de développements autour de cette technologie ont d'ores et déjà vu le jour, notamment des moteurs de rendu et des moteurs de jeux basés WebGL mais aussi des projets tel que X3DOM [BEJZ09, BJK+10]. Celui-ci rend possible la déclaration directement dans une page HTML de contenus 3D au format X3D dans n'importe quel navigateur supportant WebGL.

D'autres standards s'établissent soit de fait, soit promus par des consortiums au sein des différents domaines que réunit la réalité virtuelle. Le format Collada est par exemple devenu un standard dans le domaine des jeux vidéo, le format OGC est un standard dans le domaine des SIGs, le format U3D est un standard dans le domaine de la CAO. Néanmoins, ces standards ne répondent pas aux besoins des autres communautés, ce qui les rend inutilisables pour celles-ci. On peut voir à travers ces exemples que le succès d'un standard réside dans son extensibilité et son adéquation avec les besoins de l'ensemble des acteurs de la réalité virtuelle. Cependant les standards existants proposent la plupart du temps un consensus minimal qui écarte de fait certains besoins spécifiques et limite la réutilisabilité des contenus déjà existants encodés dans un autre format. Ces derniers ne peuvent plus être utilisés qu'en recourant à une conversion qui ne saura pas prendre en compte l'ensemble des caractéristiques du format d'origine. De plus, l'étape de conversion est souvent source d'erreurs (cf [MB08]), rendant les fichiers convertis inexploitables.

2.1.2 ENTENTE SUR UN MÉTAMODÈLE

La mise en œuvre d'un métamodèle inclut les mêmes contraintes que la création d'un standard mais elle doit également intégrer l'ensemble des fonctionnalités nécessaires aux différentes communautés de la réalité virtuelle. L'entente sur un métamodèle permet en effet l'interopérabilité en créant implicitement une correspondance entre une fonctionnalité d'un modèle et une fonctionnalité du métamodèle. Cette catégorie de solution est donc extrêmement difficile à mettre en œuvre si l'on tient compte du nombre de formats 3D existants à ce jours.

A notre connaissance, il n'existe pas de solution de ce type pour les contenus 3D mais une catégorie de format 3D répond à cette problématique : les formats d'échange. Les formats d'échange ont en effet pour vocation de permettre l'échange de contenus entre des applications dont les formats d'entrée sont incompatibles. Pour cela, ils doivent faire correspondre à chaque caractéristique d'un format A, une caractéristique dans le format d'échange E. Parmi ces formats d'échange, nous pouvons citer les formats DXF et son successeur FBX qui sont conçus pour l'échange de données entre les logiciels de modélisation de l'éditeur Autodesk. Afin de tirer parti de toutes les fonctionnalités propres aux logiciels Autodesk (notamment Maya et 3DS Max), les formats DXF et FBX ont dû les intégrer de manière à sauvegarder toutes les caractéristiques des modèles afin que celles-ci ne soient pas perdues même lorsque le logiciel importateur ne les intègre pas. Le format Collada (COLLAborative Design Activity)[AB06] est lui aussi à l'origine un format d'échange. C'est d'ailleurs cette propriété qui a rendu possible l'intégration des évolutions techniques successives du domaine des jeux vidéo et qui explique en partie sa popularité actuelle. La création de ce format avait pour objectif de répondre aux multiples problèmes générés par le manque d'interopérabilité des outils de création de contenus numériques (DCC pour Digital Content Creation) dans l'industrie du divertissement. La création de contenus numériques fait en effet appel à plusieurs outils aux formats propriétaires. Ils sont utilisés successivement au cours du processus de création donnant lieu à un pipeline de création de contenus. Collada a permis de simplifier les tâches d'importation de ces différents outils en spécifiant un modèle d'échange commun.

Les travaux de Jovanova et Preda [JP10] sur l'interopérabilité des avatars entrent dans cette catégorie de solution. Ils proposent un métamodèle pour décrire les avatars et ainsi transférer leurs apparences d'un monde à un autre. Le caractère anthropomorphe des avatars fait que ceux-ci se prêtent bien à la mise en place d'un métamodèle mais ce modèle ne peut pas s'appliquer ou être étendu à tous types de contenus 3D.

Ces exemples ne répondent pas au cadre fixé par notre étude puisqu'ils ne recouvrent pas l'ensemble des besoins de la communauté de la réalité virtuelle. En effet, ils ne préservent pas l'intégrité des formats existants et leurs utilisations impliquent une conversion vers l'un de ces standards.

2.1.3 RÉCONCILIATION DES MODÈLES

Les solutions du type réconciliation des modèles pour les contenus 3D cherchent à prendre en compte chaque format en rendant possible son intégration à différents environnements virtuels sans modification préalable de celui-ci. Bilasco et al [BVOGM07] proposent une adaptation des données 3D pour répondre à ce problème. Grâce au cadre d'application appelé "3D Application Framework" ou "3DAF" qui fournit une couche d'abstraction entre le client et la représentation des données, les auteurs adaptent une scène 3D à l'aide d'un ensemble de règles définies en fonction du type d'adaptation choisi. Le but de cette adaptation est de faire en sorte que les données 3D soient visualisables sur différents supports dont les capacités ne permettraient pas de charger la scène initiale. Ce processus est fait en amont de son déploiement dans une application ce qui ne permet pas d'accéder à l'information du fichier original durant le déroulement de l'application. Cette solution ne répond donc pas à notre problématique mais nous avons repris dans nos travaux le principe d'adaptation des contenus pour les rendre compatibles avec différentes applications.

Une autre approche est proposée par McHenry et Al. dans [MOM+11]. Ils y présentent un visualiseur universel en ligne. Il s'agit d'un service de conversion faisant appel à différents com-

posants tiers pour effectuer les conversions demandées. Le service nommé Polyglot est extensible et permet en théorie n'importe quelle conversion. Une API est également développée pour interfacer ce service avec une application quelconque afin de convertir les fichiers en sortie ou en entrée. Cette solution permet de réutiliser tous les contenus existants et offre une grande souplesse d'utilisation. Cependant elle se base sur des processus de conversion pour rendre les contenus interopérables ce qui ne répond pas aux exigences que nous nous sommes fixées. La conversion est en effet source d'erreurs et de perte d'informations.

2.2 L'INTEROPÉRABILITÉ DES COMPOSANTS LOGICIELS DE RENDU

Nous présentons ici les solutions qui existent pour rendre interopérables les composants logiciels de rendu. Il s'agit de solutions permettant d'utiliser différents composants de rendu conjointement et de les permuter sans que cela nécessite de modifier l'environnement virtuel les intégrant.

2.2.1 ENTENTE SUR UN MODÈLE

On trouve dans la littérature peu de solutions du type entente sur un modèle. Pourtant, pour les composants de rendu graphique, cela serait relativement facile à mettre en œuvre techniquement. Tout d'abord, du point de vue matériel, la diversité des cartes graphiques est moindre car ce marché se partage uniquement entre deux constructeurs : AMD [1] et NVidia [2]. De plus, tous les composants de rendu graphique se basent sur les deux bibliothèques bas-niveau existantes : OpenGL et DirectX. Ce contexte serait très favorable à la mise en place d'un moteur de rendu standard mais ce n'est pas le cas aujourd'hui. La finalité des applications, le type de rendu visuel souhaité ainsi que les évolutions techniques et de capacité du matériel sont autant de raisons qui expliquent le grand nombre de composants de rendu.

Nous avons cependant recensé une initiative qui date de 1987. Ces travaux, proposés par Beier [Bei97], présentent un noyau de rendu graphique générique et extensible appelé Generic-3D. Il se présente sous la forme d'une bibliothèque en C++ qui sert de base pour créer différents noyaux graphiques personnalisables. Elle prévoit à travers plusieurs modules la gestion des modèles, de leur rendu mais aussi des interactions. Nous pouvons également citer le standard ISO CGRM (Computer Graphics Reference Model) et plus précisément le standard GKS 3D (pour Graphical Kernel System 3D) initié au milieu des années 80 et standardisé en 1992 qui définit une interface commune pour les logiciels d'informatique graphique. Ces deux initiatives n'ont eu que très peu d'impact car elles ont été rapidement dépassées du fait des évolutions techniques au cours des années suivantes qui les ont rendues obsolètes.

2.2.2 ENTENTE SUR UN MÉTAMODÈLE

A notre connaissance, il n'existe pas de solution de ce type pour l'interopérabilité des composants de rendu. Comme pour les contenus 3D, ce type de solution est trop complexe à mettre en place dans le domaine de la réalité virtuelle, ceci étant dû encore une fois à la diversité des domaines d'application qu'elle regroupe.

2.2.3 RÉCONCILIATION DES MODÈLES

L'une des premières initiatives de ce type est due à Hinrichs en 2000 [Hin00] qui propose un graphe de scène généralisé afin de différencier clairement la spécification d'une scène et son évaluation. De cette façon, il délègue le rendu de la scène à des moteurs de rendu qui se basent ensuite sur des composants appelés "handlers" et "techniques" du graphe de scène généralisé

1. http://www.amd.com
2. http://www.nvidia.fr

pour interpréter le contenu de la scène. Il étend deux ans plus tard son concept dans [DH02] pour construire un système de rendu générique. Il s'agit d'un cadre d'application qui permet de combiner plusieurs moteurs de rendu pour la visualisation d'une même scène grâce au graphe de scène généralisé. Une implémentation du système est présentée, VRS pour Virtual Rendering System, qui permet notamment pour une scène donnée d'avoir à la fois un rendu OpenGL pour le temps réel et un rendu RenderMan pour un rendu de haute résolution. Ces travaux seront repris en 2005 dans [SRH05] pour créer un système de logiciels de réalité virtuelle appelé VR^{2}S. C'est une solution générique et extensible qui permet de tirer parti des fonctionnalités spécifiques offertes par différents moteurs de rendu grâce notamment au concept du graphe de scène généralisé et des adaptateurs qui font le lien entre leur propre système et un système de rendu existant. Cependant, cette approche ne permet pas de communication entre les différents composants ce qui ne répond pas entièrement à notre problématique.

RTSG (Real-Time Scene Graph) [RGSS09] est une autre solution de ce type. Afin de permettre le rendu en temps réel d'une scène X3D avec la technique du lancer de rayon, il propose une architecture composée d'un gestionnaire de scène et d'un système de rendu. Cette architecture charge le graphe de scène contenu dans un fichier X3D et se charge de l'optimiser pour un rendu basé lancer de rayon. Elle est personnalisable de façon à pouvoir utiliser différents algorithmes de visualisation et différents moteurs de rendu. Cette solution est très flexible mais là encore, il n'y a pas d'échanges possibles entre les différents moteurs.

Dans [RLSS10], les auteurs présentent l'architecture SORA (Service-Oriented Rendering Architecture). Il s'agit d'un architecture basée composants au sein de laquelle différents services sont en charge de certains nœuds du graphe de scène. Le parcours du graphe de scène est initialisé par l'un des services qui sait interpréter le noeud racine, ensuite lorsque celui-ci atteint un nœud qu'il ne connaît pas, il émet une requête pour ce nœud. Un composant de l'architecture va alors se charger de trouver un service compatible et se charger de la gestion de ce nœud. A travers cette architecture, les auteurs combinent plusieurs moteurs de rendu afin de pallier les manques de l'un par un autre. Cette solution est très flexible car les composants sont facilement interchangeables et elle est extensible. De plus, elle rend possible la communication entre les différents composants de rendu. La conception modulaire de SORA dans laquelle les modules sont en charge d'un ensemble de nœuds nous a influencé dans l'élaboration de notre architecture. La flexibilité et l'extensibilité de cette conception répond bien aux objectifs que nous nous sommes fixés.

Une autre solution qui ne s'adresse pas aux composants de rendu graphiques mais aux périphériques des environnements virtuels 3D est celle proposée par Taylor et al. dans [THS+01]. Il s'agit de VRPN (Virtual-Reality Peripheral Network). C'est un système qui propose une interface pour les périphériques utilisés en réalité virtuelle, cette interface est indépendante des dispositifs utilisés. VRPN se compose d'un ensemble de serveurs et d'une interface de programmation qui permet de créer des pilotes pour chaque type de périphérique. Ces pilotes permettent d'interfacer les échanges entre le dispositif utilisé (bouton, tracker, ...) et l'application cliente.

2.3 L'INTEROPÉRABILITÉ DES ENVIRONNEMENTS VIRTUELS 3D

Nous présentons dans cette partie les solutions d'interopérabilité qui prennent en compte les environnements virtuels dans leur globalité. Ces solutions ont en général pour but de permettre la mise en place d'une application cliente unique pour accéder à plusieurs environnements virtuels 3D.

2.3.1 ENTENTE SUR UN MODÈLE

La mise en place d'un standard pour permettre l'interopérabilité entre différents environnements virtuels 3D est une approche proposée de longue date. La première initiative, DIS, remonte au début des années 80 dans le projet SIMNET. SIMNET (SIMulator NETworking)[MT95] était

un projet financé par le DARPA en 1983 qui avait pour but la mise en oeuvre d'un environnement de simulation enrichi pour l'entraînement à des tâches collectives complexes. Ce projet aboutit en 1993, par la définition d'un protocole de communication à travers lequel le simulateur transmettait des informations essentielles : DIS (Distributed Interactive Simulation). Une évolution de ce protocole ainsi qu'un environnement de développement, baptisé HLA (High Level Architecture)[KWD99], ont été proposés en 1999. Le but de cet environnement de développement était de réduire les coûts de développement des systèmes de simulation et d'améliorer leurs capacités en facilitant la réutilisation et l'interopérabilité des simulateurs les composant.

Cette approche est toujours d'actualité comme le prouvent les travaux autour du standard MPEG-V [Gel08]. Ce groupe de normalisation vise à standardiser les communications entre les mondes virtuels eux-mêmes et les mondes virtuels et le monde réel. C'est une initiative globale qui cherche à définir un format intermédiaire et des protocoles pour des échanges entres mondes virtuels mais aussi avec le monde physique.

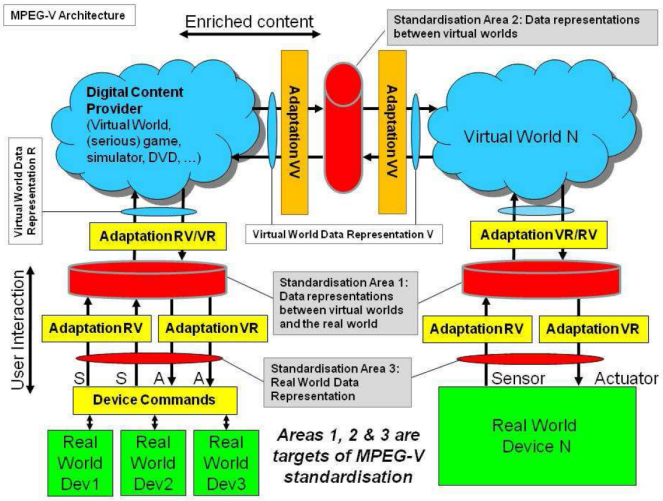


FIGURE 2.2 – *L'architecture de MPEG-V*

Une initiative similaire, VWRAP (Virtual World Region Agent Protocol)[BDL10] ne s'intéresse quant à elle qu'aux mondes virtuels du type de Second Life, c'est-à-dire des mondes divisés en régions hébergées sur des serveurs, avec des agents pour représenter les utilisateurs et un contrôle utilisateurs fait à travers une application cliente. C'est un groupe de travail lancé en 2009 et qui a pour objectif de standardiser les services et les protocoles utilisés par les environnements virtuels 3D collaboratifs. Ce groupe s'occupe d'interopérabilité pour une famille de mondes virtuels présents et futurs en se basant sur un modèle de données commun et une sémantique opérationnelle similaire. Ce groupe est composé de membres de Linden Lab, OpenSim, IBM et l'IETF. Il propose à cet fin Hypergrid [Lop11], un protocole ouvert et une architecture qui permet dans ce type d'environnement virtuel d'échanger des biens et aux utilisateurs de se téléporter d'un monde à l'autre tout en préservant leur identité numérique et leur représentation visuelle. Ce protocole a été mis en place dans le projet OpenSimulator [3] ainsi que dans son concurrent SimianGrid [4].

Pour mettre en place ces solutions pour un ensemble d'environnements virtuels, ceux-ci devront utiliser le format d'échange et les protocoles proposés par le standard. Cette nécessité n'est pas trop contraignante pour les environnements virtuels en devenir mais ceux existant devront modifier leurs contenus en les convertissant ou en les annotant afin d'être conformes au standard. Cette solution ne répond donc pas aux objectifs que nous nous sommes fixés de réutilisation des contenus 3D.

3. http://opensimulator.org
4. http://code.google.com/p/openmetaverse/wiki/SimianGrid

2.3.2 ENTENTE SUR UN MÉTAMODÈLE

Nous n'avons pas trouvé de solution de ce type pour les environnements virtuels 3D. Cela s'explique probablement par le fait qu'une telle solution aurait peu de chance d'être adoptée. En effet, cette solution, en plus d'être extrêmement compliquée à mettre en œuvre du fait des multiples techniques existantes, se heurterait à des problèmes de droits des solutions propriétaires. Elle n'intéresserait de fait que les environnements virtuels n'ayant pas de modèle économique associé.

2.3.3 RÉCONCILIATION DES MODÈLES

La réconciliation des modèles pour les environnements virtuels 3D est une autre option qui permet de créer un client unique pour plusieurs environnements virtuels. Les travaux de Soto et Allongue [SA02, SA97] s'inscrivent dans cette démarche. Ils reposent sur l'application du modèle influence/réaction issu des systèmes multi-agents pour transférer une entité d'un monde virtuel vers un autre. Une entité est un objet du monde composé d'une apparence, de composants multimédia et de comportements spécifiques. Chaque entité du monde se voit attribuer un ensemble d'attributs et d'actions au sein de son monde qui décrivent son apparence et son comportement. Lorsque l'on transpose cet objet dans un autre monde, on y transpose également ses actions de façon à prédire comment une entité et son monde hôte vont interagir. Pour s'assurer que l'ensemble des propriétés de l'entité est compatible avec le monde hôte, on calcule leur compatibilité grâce à la méthode proposée par Hubert Le Van Gong [LVG96]. Lorsque c'est le cas, on recourt à des concepts d'apprentissage artificiel et d'ontologie pour adapter les propriétés d'une entité virtuelle dans son nouveau monde. C'est une approche purement sémantique qui permet une réelle interopérabilité entre les contenus des environnements virtuels 3D. Elle tient compte en effet de toutes les propriétés de ces contenus et prévoit une adaptation de ces propriétés dans le monde hôte. Cependant, cette méthode nécessite une annotation préalable des contenus à transférer la rendant difficile à mettre en œuvre pour le transfert d'un grand nombre d'objets.

Une autre initiative, appelée PLUG [HJ08], suit également cet objectif. Les auteurs y présentent un client universel pour les environnements virtuels inspiré des clients de messagerie instantanée. Celui-ci accède aux différents mondes virtuels en se connectant à des machines hôtes sur lesquelles l'application est exécutée. Elle est ensuite retransmise au client grâce à des techniques de rendu distant. Cette solution est axée sur l'utilisateur et tente de faciliter et de simplifier son accès à un grand nombre de mondes virtuels au travers d'une interface conviviale. C'est une solution bien adaptée aux environnements virtuels en ligne mais elle ne se prête pas aux autres types d'environnement virtuel. De plus, elle ne permet pas d'utiliser d'autres composants de rendu que ceux prévus par l'application.

D'autres solutions tentent d'utiliser les navigateurs Web comme client universel comme par exemple [KCS11] qui à travers un module d'extension (ou *plugin*) utilisant le moteur Unity 3D permet le visualiser un monde virtuel. Cependant la quasi totalité de ces travaux ne s'intéressent qu'aux mondes du type de Second Life et ne sont donc pas conformes au cadre que nous nous sommes fixé.

2.4 SYNTHÈSE ET CONCLUSION

2.4.1 SYNTHÈSE

L'étude de la littérature nous a permis de compléter les caractéristiques d'une solution d'interopérabilité satisfaisante, fixées par notre cadre d'étude. Nous avons en effet essayé de tenir compte des défaillances des initiatives similaires précédentes tout en nous inspirant de beaucoup d'entre elles. Cette étude nous a également permis de confirmer le bien-fondé de ce cadre, justifiant ainsi l'intérêt de ces travaux. Du point de vue des contenus, il semble impératif de conserver l'intégrité du fichier c'est-à-dire conserver les caractéristiques multiples des modèles encodés

mais aussi préserver les fonctionnalités spécifiques des formats. Pour cela, il est nécessaire d'éviter la conversion souvent synonyme d'erreurs et de pertes de données. D'autre part, la solution proposée ne doit pas impliquer de modifier au préalable les contenus (par insertion de nouvelles données par exemple), ceci dans le but de faciliter la réutilisation de contenus. Cette démarche est en effet plébiscitée aujourd'hui comme le montre le succès des bibliothèques en ligne de modèles (TurboSquid [5], Trimble de Google [6], Mixamo [7], ...) qui suscitent un intérêt croissant aussi bien auprès des industriels que des développeurs indépendants. Par ailleurs, la solution envisagée doit permettre aux composants de rendu de collaborer sans chercher à standardiser les échanges entre les composants d'un environnement virtuel comme le nécessiterait la mise en place d'un protocole. De plus, nous souhaitons que notre solution s'applique également aux environnements virtuels déjà développés pour lesquels la mise en conformité à un standard serait très contraignante. Le tableau 2.4.1 résume ces observations. Il reprend les prérequis du cadre initial et indique pour chaque catégorie de l'étude bibliographique les nouveaux critères que celle-ci a permis d'établir.

	Caractéristiques de la solution
Cadre initial	utilisation directe des contenus
	utilisation directe des composants logiciels
	communication entre contenus et composants
	communication inter-contenus
	communication inter-composants
Contenus	préserve les fonctionnalités du format des fichiers chargés
	pas de conversion
	préserve les modèles encodés dans les fichiers chargés
Composants de rendu	collaboration possible entre eux
Environnements 3D	pas de modification préalable des contenus
	pas de mise en place de protocole

TABLE *2.2 – Les caractéristiques d'une solution d'interopérabilité satisfaisante.*

2.4.2 CONCLUSION

Au regard de ces observations, la solution qui nous paraît la plus adaptée est l'interopérabilité par réconciliation des modèles. Ce type de solution permet de prendre en compte les hétérogénéités des environnements virtuels 3D et de les préserver. Elle répond également à la plupart des caractéristiques listées ci-dessus en n'altérant pas les fichiers des contenus et en évitant les conversions. Dans cette optique, nous allons dans le chapitre suivant analyser les éléments qui constituent les environnements virtuels 3D afin d'identifier une voie de réconciliation préalable à une solution d'interopérabilité par réconciliation des modèles.

5. http://www.turbosquid.com
6. http://sketchup.google.com/3dwarehouse
7. http://www.mixamo.com/

Chapitre 3

ANALYSE DES ÉLÉMENTS CONSTITUTIFS D'UN ENVIRONNEMENT VIRTUEL 3D

DANS le chapitre précédent, nous avons étudié et classé les solutions d'interopérabilité existantes pour les environnements virtuels 3D. Cette étude nous a permis de conclure qu'une solution du type réconciliation des modèles est la solution la plus à même de répondre à notre problématique tout en respectant le cadre d'étude que nous nous sommes fixé. Ce type de solution nécessite d'identifier un canal de réconciliation entre les modèles à rendre interopérables. C'est pourquoi dans ce chapitre nous allons analyser les éléments qui constituent un environnement virtuel 3D afin d'en extraire un élément commun susceptible de servir de canal de réconciliation. Cette analyse a aussi pour but de recenser l'ensemble des éléments qu'il est essentiel de préserver et de prendre en compte pour s'assurer de la non-altération des modèles par notre solution.

3.1 ANALYSE DES CONTENUS DES ENVIRONNEMENTS VIRTUELS 3D

Pour l'ensemble des environnements virtuels présentés dans la partie 1.2, les éléments essentiels qui les constituent sont les contenus 3D. La réalisation de ces contenus est une tâche majeure de la création d'un environnement virtuel, c'est en effet à travers ces contenus que l'utilisateur va percevoir le niveau de réalisme et l'étendue qui caractérisent le monde qu'il va explorer. Nous verrons tout d'abord l'ensemble des caractéristiques que peuvent avoir les contenus des environnements virtuels 3D puis nous ferons un tour d'horizon des formats 3D utilisés pour encoder ces contenus. Nous verrons ensuite quelle structure commune est utilisée pour la majorité de ces contenus.

3.1.1 LA DESCRIPTION DES CONTENUS 3D

Les contenus des environnements virtuels 3D contiennent principalement trois types d'informations : des informations pour décrire des modèles 3D, des informations sur la scène qui englobe ces modèles et des informations sur l'animation et l'interactivité de la scène.

La description d'un modèle 3D se fait au travers de plusieurs paramètres qui définissent l'unicité de la représentation de cet objet dans l'environnement virtuel ciblé :

- **la géométrie** : Ce paramètre décrit la forme de l'objet, celle-ci peut être une forme géométrique parmi des primitives basiques (sphère, cube, cône, cylindre, ...), une combinaison de ces primitives ou alors une forme plus complexe. Un même objet peut ainsi être décrit de plusieurs façons, cela dépend de la technique et des outils de modélisation utilisés pour créer l'objet (modélisation paramétrique, procédurale, basée primitive, géométrie de construction de solide, ...). Cependant la plupart de ces outils et techniques gé-

nèrent une représentation sous forme de maillages surfaciques. Les autres représentations (volumique[Mea82], basée points[RL00, PZVBG00]) relèvent d'un usage moins courant.

- **l'aspect** : Deux paramètres permettent de définir l'aspect de l'objet : le matériau et la texture. Le *matériau* définit l'aspect de la surface de l'objet c'est-à-dire sa couleur, sa brillance, sa transparence et son interaction avec la lumière incidente. La *texture* est une image 2D, 3D ou une vidéo qui est plaquée sur certaines ou toutes les faces de l'objet. Il existe une autre technique qui permet de modifier l'aspect de la surface d'un objet, il s'agit des "*nuanceurs*" (en anglais : *shader*). Ce procédé permet d'ajouter des effets sur une face sans alourdir la représentation de l'objet afin d'accélérer son rendu. Ils exploitent des fonctionnalités des cartes graphiques permettant de modifier par programmation le processus de rendu classique.
- **la vision** : Cette catégorie de paramètres définit la façon dont l'objet va être vu par la caméra, il peut par exemple être vu constamment de face, quelque soit le placement de la caméra ("billboard") ou alors présenter des faces différentes lorsque la caméra se déplace autour de lui ou lorsqu'on le fait tourner.
- **l'affichage (performance)** : Cette catégorie de paramètres définit la façon dont va s'afficher l'objet, il s'agit en général d'améliorer les performances de rendu en proposant un affichage dégradé de l'objet comme avec la technique des niveaux de détails (*Level Of Details* ou *LOD*).

Il est possible de décrire également les propriétés de l'environnement qui englobe les objets 3D c'est-à-dire la scène. Les propriétés qui décrivent la scène sont :

- **le point de vue** : Ce paramètre permet de définir le placement de la caméra pour fixer l'angle de vision de la scène.
- **la navigation** : Ce paramètre permet de définir la façon dont la caméra va se déplacer dans la scène : par exemple, elle peut survoler les objets de la scène, tourner autour, se déplacer au niveau du sol à la manière d'un piéton.
- **l'éclairage** : Ce paramètre décrit la façon dont la scène va être illuminée à partir des trois composantes de l'éclairage : diffus, spéculaire et ambiant. On peut également définir des sources lumineuses en se basant sur les trois types de sources lumineuses en image de synthèse : la lumière multi-directionnelle (*point light*), la lumière faisceau (*spot light*) et la lumière directionnelle (*directional light*).
- **le partitionnement** : Ce paramètre définit un découpage de la scène qui permet d'optimiser certains traitements de rendu. Il existe entre autres des partitionnements sous forme de grilles 2D ou 3D (BSP (*Binary Space Partitionning*), arbre octal (*octree*), ...) ou des partitionnements de la scène liés à la visibilité (PVS (*Potential Visible Set*), portails ([TS91, LG95])).
- **autres** : une texture pour le sol, une ou plusieurs textures pour l'horizon, une piste audio ou encore des hyperliens.

La description d'un objet 3D ne s'arrête pas à la définition de sa représentation, on peut dans certains cas lui attribuer des propriétés d'interactivité, définir son comportement à la suite de l'occurrence d'un évènement ainsi que la façon dont il peut évoluer au cours du temps.

- **les événements** : Il est parfois possible de définir des capteurs d'événements sur des objets de la scène : cela permet, une fois l'événement capté, de modifier tout ou partie de la scène suivant le type de l'événement. L'évènement peut avoir été déclenché à la suite d'une action de l'utilisateur ou bien par un autre objet de la scène ou encore l'application.
- **les comportements** : Afin de rendre la scène interactive, nous pouvons définir le comportement d'un objet à la suite d'un événement. Le comportement définit alors l'ensemble des modifications apportées à l'objet. Il existe plusieurs modèles pour décrire les comportements (modèle distribué [Rey87], modèle déclaratif [DR03], ...) et certains permettent de préciser un enchaînement d'événements.
- **les articulations** : Les articulations sont des liaisons entre des solides rigides indéformables qui sont utilisées pour décrire des objets articulés. Cela permet de décrire un bras robotique ou encore un humanoïde de synthèse par exemple. La déclaration de ces liaisons permet de contrôler les mouvements de ces objets dans l'espace en définissant les degrés de liberté de chacune des liaisons.
- **les animations** : Il est également possible de décrire la façon dont un objet va être animé

au cours du temps. La description de cette animation dépend de la technique d'animation utilisée (animation par images successives ou *sprite animation*, animation par sommets ou encore animation de surface).

3.1.2 LES FORMATS 3D

Les formats 3D vont permettre de modéliser et de sauvegarder les contenus 3D. On recense plus de 140 formats 3D [MB08]. Cette variété, bien qu'importante, n'est pour autant pas figée car de nouveaux formats sont créés pratiquement chaque année. L'un des derniers en date est XML3D [SKR+10] publié en 2010.

Pour expliquer cette multiplicité, il faut se replacer dans le contexte historique des débuts de l'informatique graphique. A cette époque, les ressources nécessaires à l'exécution de programmes 3D étant très importantes, ceux-ci ne s'exécutaient que sur des stations de travails dédiées. Chaque type de station de travail avait son propre format mis au point par le constructeur. Ensuite, lorsque les capacités et la puissance de calcul des ordinateurs personnels ont permis d'exécuter des logiciels de modélisation et de visualiser leur création, le nombre de formats a encore augmenté. Chaque logiciel avait en effet son propre format propriétaire. Chacun d'eux proposait un ensemble de fonctionnalités uniques et supportait un nombre de formats limité, ce qui entraîna les premiers problèmes de compatibilité et de conversion. Pour y répondre, des formats d'échanges ont été créé mais ceux-ci ne pouvaient pas couvrir l'ensemble des fonctionnalités existantes. La situation au début des années 90 était déjà suffisamment contraignante pour que plusieurs initiatives soient proposées afin de pallier les problèmes générés par la multiplicité des formats 3D telle que le format OFF [Ros89], le format P3D [WNA90] ou encore les travaux de Koegel [Koe92]. La situation est réellement devenue inextricable lors de la mise sur le marché de la première carte graphique 3D grand public en 1996. La variété de formats 3D n'a cessé d'augmenter depuis lors et les initiatives successives pour proposer un format standardisé n'ont pas réussi à rallier tous les acteurs de l'informatique graphique. La figure 3.1 montre les dates de création des formats parmi les plus utilisés actuellement ainsi que leur durée de vie. On voit que certains ont plus de 30 ans d'existence mais sont toujours utilisés alors que de nouveaux continuent d'être créés. Nous les avons classés en 4 groupes :

- les formats issus de normes produites par un consortium (VRML, X3D, BIFS, U3D),
- les formats internes (max, maya, 3ds, obj, blender),
- les formats d'entrée des moteurs de rendu et de jeux (mesh, Unity),
- les formats d'échanges (FBX, Collada, DXF).

Les formats les plus anciens (Object3D ou OBJ, DXF, Blender ou 3DS) sont des formats qui ont été créés pour les besoins d'un outil particulier. Ce sont tous des formats propriétaires à l'exception de OBJ qui d'ailleurs sert aujourd'hui essentiellement de format d'échange. DXF a quant à lui été conçu comme un format d'échange propriétaire par AutoDesk pour pouvoir échanger des fichiers entre les différents outils d'AutoDesk. VRML permet l'affichage de contenus 3D sur le web. Son successeur, X3D, propose une extension de VRML dans un format basé XML. Collada est un format crée par Sony pour stocker ses modèles 3D. C'est donc à l'origine un format d'échange mais il est maintenant utilisé tel quel notamment dans l'industrie du jeu vidéo. Universal 3D (U3D) [CW12] est un format mis en place par un consortium d'industriels et destiné à faciliter l'échange de données industrielles. 3D Markup Language for Web (3DMLW) est un langage ouvert pour créer des contenus 2D et 3D sur le web. Crée en 2009, il fonctionne sur la plupart des navigateurs web via un module d'extension (ou *plugin*). XML3D est encore un autre format sorti en 2010 pour intégrer de la 3D dans une page web à condition d'utiliser un navigateur modifié pour fonctionner avec ce format.

Le tableau 3.1 compare les fonctionnalités offertes par différents formats 3D selon les caractéristiques de [PBSB08].

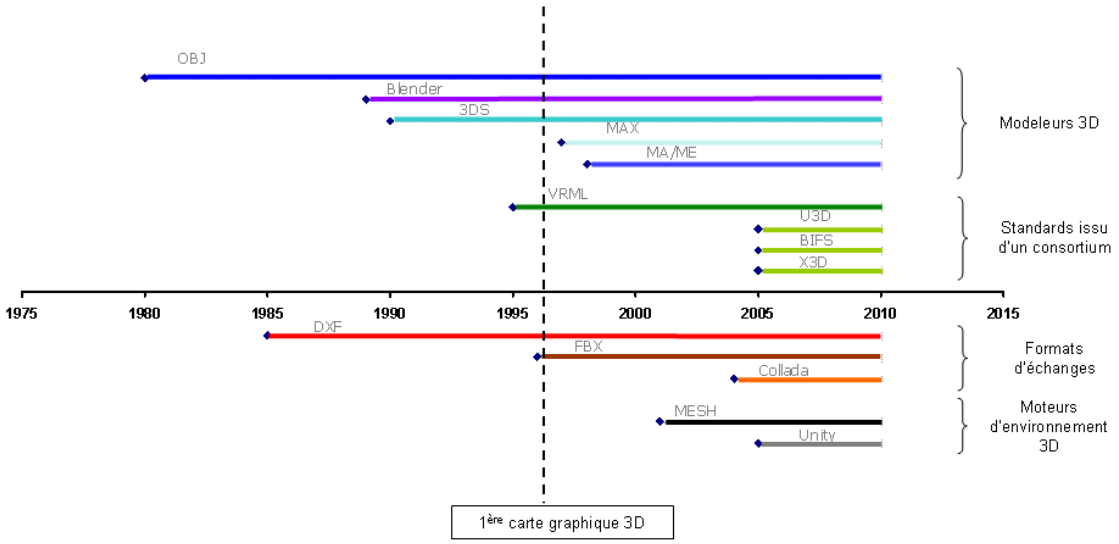


FIGURE *3.1* – *Ligne de temps des formats 3D*

TABLE 3.1: Comparaison des fonctionnalités des formats 3D.

			OBJ [1]	DXF [2]	3DS [3]	VRML [4]	FBX [5]	MAX [6]	MESH [7]	Collada [8]	X3D [9]	U3D [10]	3DMLW [11]
GÉNÉRAL		**Graphe de scène**	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✗
		Type	ASCII	ASCII/Binaire	Binaire	ASCII	ASCII/Binaire	Binaire	binaire	ASCII	ASCII	ASCII	ASCII
		Platforme	multi	multi	multi	multi	multi	multi	multi	multi	multi	multi	multi
		Maintenu	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓
		Date création	1980	1985	1990	1995	1996	1997	2001	2004	2005	2005	2009
		Licence	ouvert	propriétaire	propriétaire	standard ouvert	propriétaire	propriétaire	MIT	SCEA	standard ouvert	standard ouvert	GPL
		Extension	.obj	.dxf	.3ds	.wrl	.fbx	.ma	.mesh	.dae	.x3d	.u3d	.3dmlw
		Applications	Export de modeleurs, Second Life	AutoCAD	3DS Max	Web, ...	format d'échange entre les logiciels Autodesk	3DS Max	Ogre, Irrlicht, ...	Echange, export	Web, industrie, ...	CAO	Web
DESCRIPTION	*Scène*	**Navigation**	✗	✗	✗	✓	✗	✗	✗	✗	✓	✗	✗
		Point de vue	✗	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓
		Environnement	✗	✗	✗	✓	✓	✗	✗	✗	✓	✗	✗
		Éclairage	✗	✗	✓	✓	✓	✓	✗	✓	✓	✓	✗
		Audio	✗	✗	✗	✓	✓	✗	✗	✗	✓	✗	✓

1. [forf]
2. [forc]
3. [fora]
4. [CB97]
5. [ford]
6. [forb]
7. [fore]
8. [AB06]
9. [for06]
10. [Sta07]
11. [3D 09]

. . . continued

			OBJ	DXF	3DS	VRML	FBX	MAX	MESH	Collada	X3D	U3D	3DMLW
	Objet	**Géométries**	polygones (avec normales), courbes	ligne polygones de 3 ou 4 sommets polyligne	triangle (sans normales)	primitives, extrusion, polygones, lignes, points, grilles d'élévation, textes, NURBS	polygones, NURBS		polygones	courbes, polygones, NURBS	primitives, extrusion, polygones, lignes, points, grilles d'élévation, textes, NURBS		primitives, imports (.3ds, .obj, .an8 et .blend)
		Shaders	✗	✗	✓	✗	✓	✓	✓	✓	✓	✓	✓
		Matériaux	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓
		Métadonnées	✗	✓	✗	✗	✗	✗	✗	✓	✓	✓	✗
		Textures	✓	✗	✓	✓	✓	✓	✗	✓	✓	✓	✓
		Performance	✗	✗	✗	✓	✗	✗	✓	✓	✓	✓	✗
INTERACTION		**Animation**	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✗
		Évènements	✗	✗	✗	✓	✗	✗	✗	✗	✓	✗	✓
		Scripts	✗	✗	✗	✓	✗	✓	✗	✗	✓	✗	✓
		Comporte-ments	✗	✗	✗	✓	✗	✗	✗	✗	✓	✓	✗
		Physique	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗
		Articulations	✗	✓	✓	✗	✓	✓	✓	✓	✓	✗	✗

3.1.3 La structuration des représentations

De façon à rendre le traitement de la scène plus efficace, il est utile de structurer sa représentation en regroupant des éléments de la scène en fonction de critères. On distingue trois critères de regroupement qui peuvent cohabiter au sein d'une même scène ou d'un même modèle :

- la structuration logique : selon l'application visée, on décompose la scène en sous-ensembles liés logiquement entre eux. On établit généralement une hiérarchie entre les différents éléments sous la forme d'un arbre. Par exemple, pour un solide articulé, il est intéressant d'organiser les éléments du solide de façon à décrire un mouvement comme la composition de mouvements élémentaires.
- la structuration de construction : elle est liée à la manière de modéliser l'objet comme dans le cas de la géométrie constructive de solides (CSG). Les opérations ensemblistes qui permettent de modéliser l'objet peuvent être décrites sous la forme d'un arbre qui organise l'ordre et la priorité de ces opérations.
- la structuration spatiale : elle permet d'organiser la scène de façon à dégager des espaces englobants qui vont permettre d'accélérer le rendu en temps réel de ces scènes. Certains modèles comme les arbres octaux possèdent implicitement cette notion de structuration de l'espace.

Il existe plusieurs façons de formaliser la représentation d'une scène 3D en fonction de ces trois critères (liste, tableau, arbre, ...). Cependant, celle adoptée par la quasi-totalité des formats de fichiers 3D est le **graphe de scène**. La scène est formalisée sous forme d'un graphe orienté acyclique (en anglais *directed acyclic graph* ou *DAG*) dont les arcs sont des relations hiérarchiques entre les objets et les noeuds sont soit des objets de la scène, soit des noeuds de structuration plus ou moins complexes (nœud de groupe, nœud de positionnement, nœud de niveau de détail (*LOD*) ou encore nœud d'activation (*switch*)). Le noeud le plus haut dans la hiérarchie est le noeud racine et chaque noeud n'a qu'un seul parent. Comme le montre le tableau 3.1, la plupart des formats de fichiers 3D utilisent une structuration en graphe de scène. Il existe d'autres structurations mais il s'agit de formats historiques comme le format OBJ défini par la société *Wavefront Technologies* en 1980.

La fréquence de cette structuration s'explique par le fait que depuis son introduction dans Open Inventor [SC92] en 1992, aucun concept aussi efficace et flexible n'a été proposé pour le rendu de la scène. Celui-ci reflète en effet les étapes du processus de rendu qui permettra de visualiser la scène. Cette structuration permet également de garder les relations de dépendances entre les différents objets de la scène (alignement, superposition, composition, attachement). Ainsi, la transformation courante d'un nœud donné s'obtient par composition des transformations sur le chemin menant de la racine à ce nœud. Ceci permet de calculer de manière plus efficace quels sont les objets hors du champ de vision. Cette structuration possède de plus l'avantage de faciliter les manipulations ; si l'on déplace un nœud, ses enfants suivront automatiquement.

3.2 Analyse des composants logiciels des environnements virtuels 3D

Dans cette partie, nous allons recenser les composants qui vont contribuer à la restitution d'une scène tridimensionnelle. Nous verrons ensuite quelles sont les contraintes qui pèsent sur le processus de rendu. Il s'agit en effet d'une tâche complexe car elle doit s'adapter au dispositif d'affichage ciblé puis gérer en temps réel l'ensemble des interactions prévues ainsi que leurs impacts dans l'environnement virtuel en fonction du degré d'immersion voulu.

3.2.1 LES COMPOSANTS DE RENDU DES ENVIRONNEMENTS VIRTUELS 3D

Le schéma 3.2 présente l'architecture standard d'un monde virtuel sous forme de modules. L'organisation de ces modules peut différer d'un environnement virtuel à l'autre mais on retrouve pour chacun d'entre eux ces même éléments. Ce schéma s'inspire en grande partie de celui proposé par Jason Gregory dans son livre "Game Engine Architecture" [GL09]. Nous allons détailler ici les fonctionnalités et le rôle de chacun des modules au sein d'un environnement virtuel.

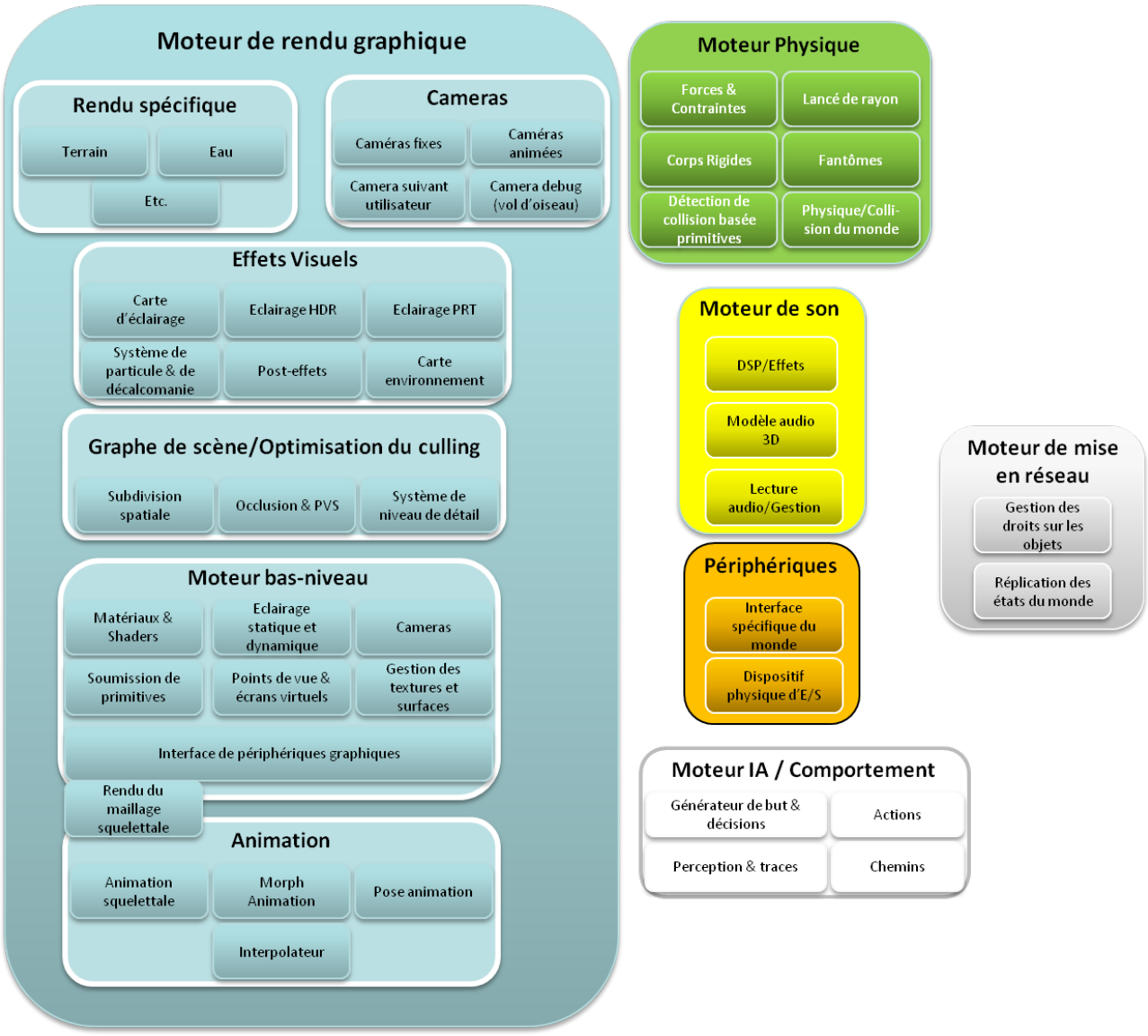


FIGURE *3.2 – Les différents composants d'un monde virtuel*

3.2.1.1 Le moteur de rendu graphique

Le moteur de rendu graphique est le composant qui gère l'affichage à l'écran de la scène 3D à partir du modèle fourni par le CPU. Ce processus nécessitant de nombreuses étapes, le moteur de rendu graphique est le composant le plus complexe. Son architecture peut être organisée de différentes façons mais celles-ci tendent à être harmonisées au fil du temps. En effet, le rendu est

extrêmement lié aux composants matériels graphiques sur lesquels il va s'exécuter. Le marché des cartes graphiques ne se partageant aujourd'hui plus qu'entre deux protagonistes, *NVidia* et *ATI*, le matériel graphique est de plus en plus uniforme et les moteurs de rendu suivent cette tendance. De plus, quelque soit l'architecture du moteur de rendu, il reposera sur une interface de programmation graphique et comme pour les cartes graphiques, il en existe deux : *OpenGL* et *DirectX*. Le moteur de rendu graphique gère cinq modules :

- **Moteur bas-niveau** : Cette partie du moteur de rendu gère le rendu d'un ensemble de primitives géométriques le plus rapidement et le plus richement possible sans prendre en compte les parties de la scène qui sont visibles ou non. Son rôle est de transmettre la géométrie, les matériaux, les textures, les caméras et les lumières à la chaîne de rendu de la carte graphique c'est-à-dire de convertir les primitives de la scène en données interprétables par le matériel graphique. Pour cela, elle s'appuie sur l'interface de périphériques graphiques qui gère les instructions de bases de la bibliothèque graphique utilisée, *OpenGL* ou *DirectX*.
- **Graphe de scène/Optimisation de l'élimination des parties cachées** : C'est au sein de cette partie que le graphe de scène est élagué afin de n'afficher que ce qui est visible du point de vue de la caméra. Les différentes techniques mise en œuvre lors de cette étape sont présentées dans la partie 3.2.3.1.
- **Effets visuels** : Cette partie se développe à mesure que de nouveaux effets visuels sont disponibles ; cartes d'éclairage, cartes d'environnement, éclairage PRT ou encore éclairage HDR.
- **Animation** : Cette partie gère l'animation aussi bien des avatars que des objets animés du monde. L'animation étant une tâche complexe, des bibliothèques d'animation ont été développées et sont maintenant très couramment utilisées par les environnements virtuels actuels. Les plus populaires sont *Granny 3D* [12], *Havok Animation* [13], *Edge* [14] ou encore *Mecanim* [15].
- **Rendus spécifiques et caméras** : Cette partie gère des modules spécifiques de rendu nécessaires pour certains environnements. L'eau par exemple nécessite généralement un module de rendu propre. Il existe en effet de nombreux types d'eau : océan, piscine, rivière, chute d'eau, fontaine, jet d'eau, flaque, ... Pour chacun des types énoncés, il faut une technique de rendu spécifique. A cela s'ajoute le fait qu'une étendue d'eau est la plupart du temps animée d'un mouvement (vague, courant, ...) et requière donc la mise en place de pavage (tesselation) dynamique et de niveaux de détail afin d'optimiser son rendu sans surcharger l'unité de rendu graphique. Une autre raison pour la mise en œuvre d'un module spécifique est la gestion des interactions entre l'eau et les objets du monde (par exemple le fait de faire flotter un objet). Le rendu de l'eau est pour toutes ces raisons une combinaison de plusieurs techniques de rendu et de sous-systèmes. Par exemple, pour une chute d'eau, on aura un shader spécialisé ainsi qu'un système de particules pour la brume mais aussi une sorte de système de calque pour l'écume. La gestion des différentes caméras est aussi souvent prise en charge par le moteur de rendu, ce n'est cependant pas toujours le cas. C'est une tâche assez complexe car cela consiste à maintenir le point de vue tout en gérant les collisions avec les objets rigides du monde et éviter les occultations entre la caméra et sa cible.

3.2.1.2 Le moteur physique

Le moteur physique est le module qui gère la simulation des règles physiques de l'environnement virtuel. Selon le niveau de réalisme voulu dans le monde, des lois physiques du monde réel vont être simulées dans le monde virtuel. Ainsi, le moteur physique va gérer la réaction liée à la collision entre deux corps rigides en empêchant ceux-ci de passer l'un au travers de l'autre par exemple. Ce composant est d'autant plus complexe que les lois physiques simulées sont nom-

12. http://www.radgametools.com/granny.html
13. http://www.havok.com/index.php?page=havok-animation
14. http://edgeanimation.com/
15. http://unity3d.com/unity4/

breuses. Là encore, des moteurs physiques standards sont de plus en plus utilisés par les mondes virtuels. Parmi ceux-là nous pouvons citer *PhysX* [16] de NVidia, *ODE* [17], *Havok Physics* [18], *Bullet* [19] ou encore *Tokamak* [20].

3.2.1.3 Le moteur de comportement et d'intelligence artificielle

Le moteur de comportement et d'intelligence artificielle est une autre brique complexe d'un environnement virtuel. Il se charge de traiter le comportement des êtres animés du monde autre que les avatars des utilisateurs. Il va notamment contrôler le comportement d'un être lors d'une interaction avec un avatar afin que ce comportement s'accorde avec le contexte du monde. Par exemple, il va déclencher un comportement d'attaque si cet être est un ennemi dans un jeu du type FPS. Il gère également une tâche appelée recherche de chemin (*path finding*) pour permettre un déplacement autonome et optimal des avatars dans le monde virtuel. Pour ce module encore, il existe des environnements de développement qui évitent aux concepteurs de monde virtuel de refaire ce module pour chaque nouveau monde développé. Le moteur de comportement le plus utilisé est *Havok Behavior* [21]. Il existe une offre plus large d'environnements de développement pour l'intelligence artificielle parmi lesquels *Autodesk Kynapse* [22] et *Havok AI* [23].

3.2.1.4 Le moteur de son

Le moteur de son est le module qui se charge des sons de l'environnement virtuel. Il permet de jouer dans le monde différentes sources audio et de les spatialiser ; c'est-à-dire de les diffuser en tenant compte de la position du récepteur dans le monde. Par exemple, s'il y a une fontaine dans le monde virtuel qui émet un bruit d'eau, le son perçu par l'utilisateur va augmenter à mesure que celui-ci s'approche de la fontaine. De plus, selon l'orientation de la caméra par rapport à l'objet émetteur de son, le son sera plus audible à droite qu'à gauche ou inversement.
Microsoft propose un ensemble d'outils audio pour la plateforme DirectX appelé *XACT* [24]. Electronic Arts a développé son propre outil appelé *SoundR !OT*. Sony a fait de même avec un moteur audio 3D du nom de *Scream*. Il existe également des outils très populaires et indépendants tels que *Fmod* [25] ou encore la bibliothèque *OpenAL* [26].

3.2.1.5 Le moteur de mise en réseau

Un environnement virtuel multiutilisateur est une application en ligne, il a donc besoin d'un module qui gère cette caractéristique. Celui-ci a essentiellement deux tâches : gérer la connexion simultanée de multiples utilisateurs et synchroniser l'état du monde pour chaque utilisateur connecté afin que le monde reste consistent. Il existe des moteurs dédiés tels que *Raknet* [27], *Netdog* [28] ou *Gnet* [29].

16. http://www.nvidia.com/object/physx_new.html
17. http://www.ode.org/ode.html
18. http://www.havok.com/index.php?page=havok-physics
19. http://bulletphysics.org/wordpress/
20. http://www.tokamakphysics.com/
21. http://www.havok.com/index.php?page=havok-behavior
22. http://usa.autodesk.com/adsk/servlet/pc/index?id=11390544&siteID=123112
23. http://www.havok.com/index.php?page=havok-ai
24. http://msdn.microsoft.com/en-us/library/bb174772.aspx
25. http://www.fmod.org/
26. http://connect.creativelabs.com/openal/default.aspx
27. http://www.jenkinssoftware.com/
28. http://www.pxinteractive.com/
29. http://www.gamantra.com/

3.2.1.6 Le gestionnaire de périphériques

Un monde virtuel doit gérer des entrées de l'utilisateur qui proviennent de différentes sources :
- d'un clavier et d'une souris,
- d'une manette de jeu,
- d'autres contrôleurs de jeu spécifiques tels que WiiMote, volant, tapis de danse, ...

L'environnement virtuel doit prendre en compte ces entrées mais aussi de plus en plus souvent générer une rétroaction en réponse comme par exemple le son émis par la WiiMote. Pour ce module, on a souvent recours au système VRPN[THS+01] (cf paragraphe 2.2.3) qui interface les échanges entre les périphériques et l'application cliente de l'environnement virtuel.

3.2.2 Les contraintes du rendu temps réel

La restitution de l'environnement virtuel se fait au cours du processus de rendu dont le rôle est d'afficher sur le périphérique de sortie la scène 3D composée de modèles géométriques tridimensionnels, des animations définies, d'un éclairage donné et de textures. Le processus standard (cf figure 3.3) se découpe en plusieurs étapes qui vont transformer la scène 3D vectorielle en mémoire en une image 2D.

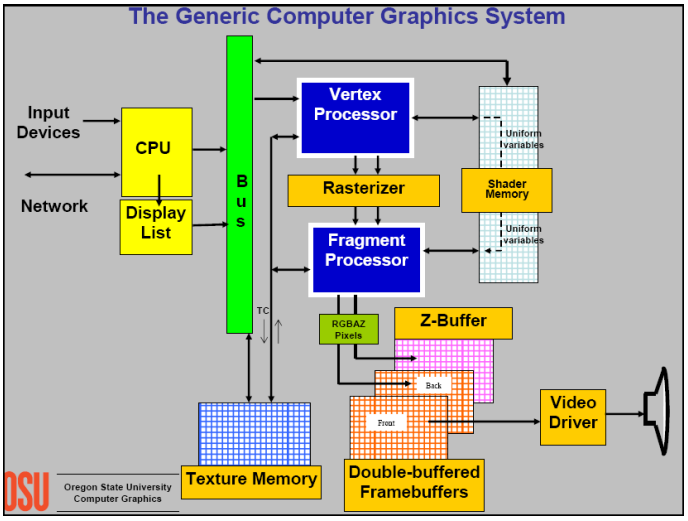


FIGURE 3.3 – *Les étapes du processus de rendu graphique[BC08]*

Comme toute application graphique en temps réel, le rendu des environnements virtuels 3D implique d'optimiser le plus possible les étapes du processus d'affichage. Ceci afin de ne pas perturber l'impression d'immersion de l'utilisateur qui passe essentiellement par le retour visuel. Pour que la scène ne paraisse pas saccadée, la fréquence de rafraîchissement doit être au minimum de 25 Hz ce qui laisse peu de temps à la machine pour effectuer le rendu d'une image à partir d'une scène 3D. Ce temps peut encore diminuer avec des images stéréoscopiques ou si l'environnement virtuel intègre une ou plusieurs interfaces fonctionnant à des fréquences plus élevées (bras à retour d'effort, capteurs de localisation, ...). Nous allons présenter dans le paragraphe suivant les principales techniques utilisées pour accélérer ce rendu.

3.2.3 Les éléments d'optimisation du processus de rendu

L'optimisation du rendu joue généralement sur deux éléments : l'élimination des parties cachées et le modèle d'éclairage.

3.2.3.1 L'élimination des parties cachées

Ce processus (en anglais *culling*) consiste à élaguer la scène afin de n'afficher que ce qui est visible du point de vue de la caméra. On supprime de la scène rendue ce qui est derrière l'observateur, ce qui est hors champs ou encore ce qui est caché derrière un autre objet. Ce dernier processus est appelé occultation. L'élimination des parties cachées permet de réduire le temps d'affichage en réduisant autant que possible le nombre d'objets affichés. Plusieurs techniques sont utilisées et combinées afin d'effectuer cette optimisation.

Une première approche est de partitionner la scène afin d'obtenir un découpage qui permettra de savoir rapidement quelles zones du découpage sont visibles ou non. On parle de **subdivision spatiale**. On divise l'espace récursivement en un ensemble d'hyperplans. On peut diviser l'espace de façon binaire, on parle alors de *subdivion binaire* (*BSP*pour *Binary Space Partitionning*). L'espace peut être divisé en *arbre octal* (*octree*). Les nœuds de l'arbre divisent l'espace en huit zones à chaque itération. Cette technique permet de déterminer facilement les zones visibles et est très efficace pour les scènes où il y a de grandes régions inoccupées. Une autre variante analogue, appelée *arbre quaternaire* (*quadtree*), divise l'espace en cellules ayant une capacité maximum, lorsque celle-ci est dépassée, la cellule est divisée de nouveau. Dernière variante, l'*arbre kD* (*KD-tree* ou *k-dimensional tree*) est un arbre binaire comme l'arbre quaternaire dans lequel les nœuds sont des points en k dimensions. Chaque nœud de l'arbre divise l'espace en deux espaces de cardinalités équivalentes.

Une seconde approche se base sur un pré-calcul des zones visibles d'un point de vue donné ou d'une région donnée. C'est celle choisie pour le *PVS* (*Potential Visible Set*) qui découpe l'espace navigable de la scène en plusieurs cellules, pour chaque cellule l'ensemble des polygones potentiellement visibles depuis celle-ci est pré-calculé. Ainsi, lorsque le point de vue est positionné dans une cellule donnée, on a rapidement une estimation des polygones visibles. Il existe une optimisation de cette technique appelée *portail* [TS91, LG95] qui permet de supprimer une grande partie des polygones du PVS. Cette technique s'applique aux scènes se déroulant en intérieur.

Une autre optimisation plus courante mais qui est définie au moment de la modélisation est la technique des *niveaux de détail* (*LOD* pour *Level Of Detail*). Afin de fluidifier le rendu, certains éléments de la scène sont affichés avec un niveau de détail plus important à mesure que la caméra se rapproche de l'objet. Un même objet existe en plusieurs définitions, c'est-à-dire avec plus ou moins de polygones pour le définir. Si la caméra est éloignée de l'objet, on affiche le modèle avec la plus faible résolution puisque sa surface de projection à l'écran est faible. A l'inverse, lorsque la caméra est très proche de l'objet, ce sera le modèle très détaillé qui sera affiché. Selon les besoins, il peut y avoir autant de niveaux que nécessaire, voire des niveaux de détails continus (cf [Hop96, LE97]).

3.2.3.2 Le modèle d'éclairage

Le calcul de l'éclairage de la scène est une étape importante du processus de rendu. Plus le niveau de réalisme souhaité est élevé et plus celui-ci prend en compte de paramètres différents, le rendant d'autant plus long et complexe. Pour optimiser ce calcul, on utilise un modèle d'éclairage simplifié proposé par Phong [**?**]. Il distingue trois types d'intensités lumineuses :

- l'intensité ambiante,
- l'intensité diffuse,
- la réflexion spéculaire.

Ce modèle définit une luminance constante pour tous les triangles de la scène. Sa complexité est proportionnelle au nombre de triangles et au nombre de sources lumineuses. Les optimisations de Gouraud [Gou71a] et Phong [Pho75] sont fréquemment utilisées pour améliorer les discontinuités d'intensité aux bords des triangles, induites par ce modèle. Afin d'optimiser ce calcul, il est courant d'utiliser un graphe de scène supplémentaire qui va organiser les objets en fonction de leurs matériaux. Ceci permet de calculer une seule fois le rendu de l'éclairage d'une surface lorsque plusieurs objets sont composés de matériaux identiques. D'autres techniques plus

récentes existent qui permettent d'obtenir un éclairage plus réaliste comme les cartes d'éclairage, les cartes d'environnement, l'éclairage PRT (*Precomputed Radiance Transfer*) ou encore éclairage HDR.

On voit que pour l'ensemble de ces techniques, la structuration de la scène 3D en graphe de scène facilite la mise en œuvre de ces optimisations. Cela explique la prédominance de ce modèle et son intégration dans la plupart des composants des environnements virtuels 3D.

3.3 SYNTHÈSE ET CONCLUSION

3.3.1 LES ÉLÉMENTS COMMUNS AUX CONTENUS 3D ET AUX COMPOSANTS DE RENDU

Nous avons recensé dans la partie 3.1.1 les principales informations que contiennent les contenus des environnements virtuels 3D. Ces informations sont ensuite prises en charge par les composants logiciels d'un environnement virtuel 3D. Chacun de ces composants prend en charge une partie de ces informations et les restitue dans le but d'enrichir l'environnement virtuel. Le tableau 3.3.1 donne pour chaque information donnée par les contenus, le ou les composants qui les prennent en compte. On observe que de part et d'autre, ce sont les mêmes informations qui sont stockées dans les contenus 3D et prises en charge par les différents composants des environnements virtuels 3D. On constate de plus que la structuration en graphes de scènes est partagée par la majorité des formats 3D ainsi que par les principaux composants logiciels des environnements virtuels. Cette structuration offre de nombreux avantages autant pour les formats que pour les composants ce qui explique la généralisation de son usage.

		Informations des contenus	**Prise en charge des composants**
Description	Navigation	marcher, voler, ...	MRG
	Point de vue	point de vue, fenêtre d'affichage	MRG
	Environnement	fond (background), nœud d'activation (switch)	MRG
	Éclairage	ambiante, directionnelle	MRG
	Audio	lien vers fichier son	MS
	Géométries	primitives, ensemble de faces indexées	MRG
	Shaders	lien vers scripts	MRG
	Matériaux	transparence, réflexion, ...	MRG
	Texture	lien vers un fichier image	MRG
	Performance	niveaux de détails, streaming, ...	MRG
Interactions	Animation	animations par squelette, ...	MRG,MA,MR
	Evénements	actions de l'utilisateur, temps	MRG, Périphériques
	Comportement	description de comportement	MA
	Physique	du monde, des objets	MP
	Articulations	description des liens	MRG,MA
Structuration	Graphe de scène	la très grande majorité des formats	MRG, MP, MA, MS
	Autre	object3D	MR

TABLE *3.2 – La gestion des informations des contenus 3D par les composants logiciels (MRG= Moteur de rendu graphique, MP= Moteur physique, MA= Moteur d'animation et d'IA, MS= Moteur de son et MR=Moteur de mise en réseau).*

3.3.2 CONCLUSION

L'analyse des contenus et des composants logiciels nous a permis d'identifier une voie de réconciliation pour notre solution d'interopérabilité : le graphe de scène. Nous avons vu que cette structuration se retrouve aussi bien dans les formats 3D qui encodent les contenus des environnements virtuels 3D que dans leurs composants logiciels. Cette structure possède en effet des propriétés qui offrent des bénéfices pour le traitement des contenus mais également pour les processus de restitution opérés par les différents composants d'un environnement virtuel 3D. Les travaux de Hinrichs et al. [Hin00, DH02, SRH05] sur le graphe de scène généralisé se basent sur le même constat. Ils nous ont donc conforté dans notre démarche. Les adaptateurs qu'ils proposent ainsi que les services décrits par Repplinger et al. [RLSS10], nous ont ensuite aidé dans la conception de notre solution. Dans la suite de ce document, nous allons présenter notre contribution qui consiste en une solution d'interopérabilité par réconciliation des modèles pour les environnements virtuels 3D.

Deuxième partie

Contribution

Chapitre 4

Du graphe de scène à l'adaptateur de graphes de scène

Au début de ce travail, nous nous sommes fixé un cadre précis pour répondre à notre problématique : rendre possible l'interopérabilité entre les contenus 3D et les composants de rendu des environnements virtuels 3D. Ce cadre définit trois caractéristiques qui nous paraissent essentielles pour la conception d'une solution satisfaisante :

- être compatible avec le plus de formats de fichiers 3D possibles, ces derniers contenant la représentation des contenus 3D,
- être compatible avec la plupart des composants de rendu, le rendu d'un environnement virtuel 3D repose en effet généralement sur plusieurs composants distincts (un moteur de rendu, un moteur physique, un moteur de comportement, ...),
- rendre possible la communication entre ces différents éléments constitutifs d'un environnement virtuel.

Pour réaliser cela, la solution que nous proposons se base sur un élément commun à la fois aux formats 3D et aux composants de rendu : *le graphe de scène*. Cette structure nous sert de canal de communication entre les différents formats de fichiers et les composants de rendu, les rendant ainsi interopérables. Nous commencerons par expliquer les raisons qui nous ont poussé vers cette approche puis nous verrons le principe fondamental sur lequel repose notre architecture : l'adaptateur de graphes de scène.

4.1 Le graphe de scène comme dénominateur commun

Bien que les formats de fichiers 3D et les composants de rendu aient des utilisations et des fonctionnalités différentes, ils présentent de nombreuses similitudes. La plupart d'entre eux se basent en effet sur une représentation sous forme de graphe de scène. L'état de l'art nous a permis de confirmer la prédominance de cette structuration pour la représentation des scènes 3D tant dans les formats de fichiers que dans les composants de rendu. Il existe quelques exceptions au sein des formats de fichiers comme par exemple le format Objet 3D (.obj) mais celles-ci sont rares et ne se rencontrent jamais pour les formats apparus après 1996. Pour les composants de rendu, l'absence de structuration en graphe de scène est également exceptionnelle. Le graphe de scène proposé ne permet pas nécessairement de créer une arborescence complexe, il se résume parfois à un noeud racine et à un seul niveau d'enfants. C'est le cas des moteurs physiques. Les similitudes ne s'arrêtent cependant pas là. L'agencement des données contenues dans le graphe de scène suit la même logique et la même organisation. On retrouve par exemple les mêmes structures de description de formes avec une séparation entre la géométrie et son apparence. On y retrouve également les mêmes transformations géométriques ainsi que les mêmes formes de base (cube, sphère, cylindre, ...).

Pour le cas des formats 3D, cela s'explique vraisemblablement par le fait que depuis son introduction dans Open Inventor[SC92] en 1992, aucun concept aussi efficace et flexible que ce dernier n'a été proposé. Il reflète en effet les étapes du processus de rendu que les données encodées vont subir.

Dans le cas des composants de rendu graphiques, l'argument est également valable mais il y a une autre explication plus triviale. En effet, ils reposent tous sur l'une des deux interfaces de programmation disponibles : OpenGL et DirectX. Ces deux interfaces de programmation utilise un mécanisme d'empilement de matrices $4x4$ pour les transformations, sur lequel s'appuie directement le graphe de scène. Ces composants proposent des interfaces de programmation qui permettent une abstraction plus intuitive pour le développeur que celles offertes par ces bibliothèques bas-niveau. C'est pourquoi leurs évolutions suivent les évolutions de ces interfaces de programmation autant que les évolutions des unités de rendu graphique (GPU). Dans la suite de ce document, nous parlerons de *graphe de scène de format* pour le graphe de scène utilisé par les formats 3D et de *graphe de scène de moteur* dans le cas des composants de rendu (cf figure 4.1).

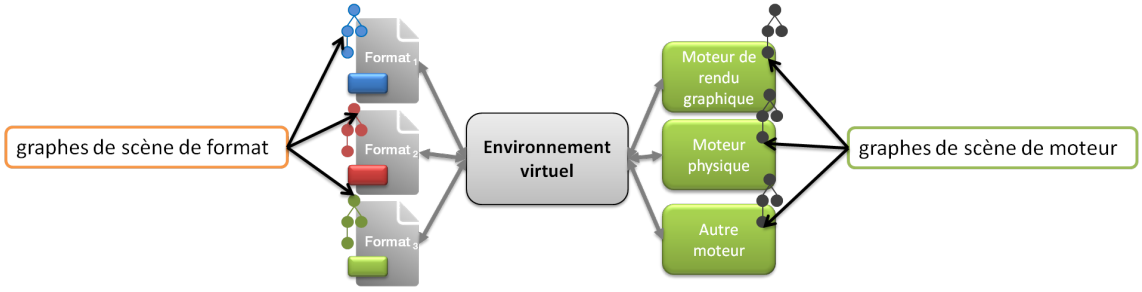


FIGURE *4.1* – *Les **graphes de scène de format** et les **graphes de scène de moteur**.*

Lors de la conception de l'architecture de notre solution, notre première idée était de s'inspirer du modèle du web qui est en effet un exemple d'interopérabilité. L'interopérabilité du Web a été favorisée par l'établissement dès sa création de standards (HTML et HTTP) par le W3C. En effet, la mise en place de ces normes dès les années 90 a permis de structurer tous les développements et ainsi d'assurer la compatibilité avec tous les navigateurs. Lorsqu'un format n'a pas été normalisé par le W3C, pour qu'un navigateur web soit capable de le prendre en charge, il suffit de fournir un module d'extension (en anglais *plugin*) pour ce format. Ce modèle nous a orienté vers une architecture modulaire dans laquelle chaque format et chaque composant de rendu est pris en charge par un module dédié. L'étude des solutions pour l'interopérabilité des composants de rendu existantes (cf paragraphe 2.2) a ensuite justifié et conforté cette approche.

Afin de faire communiquer les modules en charge de chacun d'un modèle, nous avons conçu une interface de programmation (ou API) qui utilise le graphe de scène comme canal d'échange pour réconcilier chaque modèle. Cette interface de programmation n'a pas pour objectif de proposer une représentation intermédiaire des graphes de scène visualisés dans un environnement virtuel 3D mais plutôt d'adapter la représentation des graphes de scène de format vers la représentation des graphes de scène de moteur et inversement. Cette notion d'adaptation se rapproche de celle du patron de conception adaptateur (GoF 139 [GHJV95]), c'est pourquoi nous en avons repris la terminologie et avons appelé notre architecture, l'adaptateur de graphes de scène (Scene Graph Adapter ou SGA).

4.2 L'ADAPTATEUR DE GRAPHES DE SCÈNE : PROPOSITION FONDAMENTALE

Le SGA a pour rôle d'adapter des graphes de scène de format en graphes de scène de moteur et de réaliser l'opération inverse. Cette adaptation se base sur les similitudes intrinsèques

des graphes de scène de format et des graphes de scène de moteur et va permettre de maintenir la cohérence de tous les graphes de scène impliqués tout au long du déroulement de l'application. Pour réaliser cette adaptation, il manipule des nœuds dont il transcrit les propriétés d'une représentation à une autre. Afin de clarifier le rôle du SGA, nous présentons dans cette partie un formalisme qui explicite sa fonction.

Soit Z l'ensemble de tous les formats 3D basés graphe de scène. Un contenu 3D est défini par le couple (f,z) où f est un fichier et z est un format 3D avec $z \in Z$.

D'autre part, nous considérons E l'ensemble de tous les moteurs qui constituent les composants de rendu d'un environnement virtuel 3D (moteur de rendu, moteur physique, moteur de comportement, moteur d'intelligence artificielle, ...). Un moteur peut être caractérisé par le couple (e,t) où $e \in E$ et t est son type (rendu, physique, comportement, ...) .

Prenons N, l'ensemble des nœuds dans un graphe de scène de format et N', l'ensemble des nœuds dans un graphe de scène de moteur. On considère qu'un nœud a un index unique au sein d'un graphe de scène donné. Le triplet (f,z,n_i) avec $n_i \in N$ caractérise alors le nœud d'index i dans le graphe de scène de format du fichier f encodé dans le format z. De même, le triplet (e,t,n'_i) avec $n'_i \in N'$ est le nœud d'index i dans le graphe de scène de moteur du moteur e de type t.

Nous considérons qu'un environnement virtuel 3D au cours de son exécution met en relation un ensemble de graphes de scène : les graphes de scène des fichiers sources qu'il utilise et les graphes de scène des moteurs sur lesquels reposent ses rendus. Ces triplets permettent de caractériser de façon unique un nœud d'un graphe de scène donné parmi l'ensemble de graphes de scène.

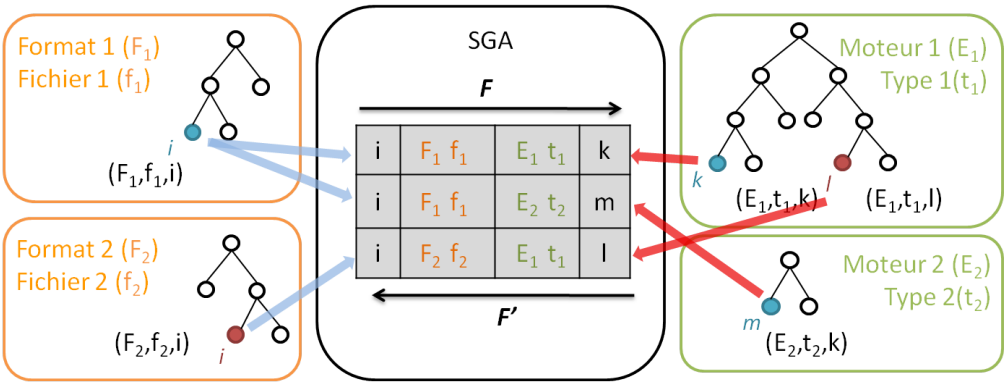


FIGURE 4.2 – *Le principe fondamental de l'adaptateur de graphes de scène.*

Le SGA permet de retrouver :

1. un nœud d'un graphe de scène de format étant donné un nœud d'un graphe de scène de moteur,
2. des nœuds de différents graphes de scène de moteur étant donné un nœud d'un graphe de scène de format.

Soit n_i un nœud d'un graphe de scène de format, le SGA renvoie $\emptyset$ ou un ensemble de nœuds issus de différents graphes de scène de moteur. F est la fonction de récupération :

$$F(f,z,n_i) = \{(e_j,t_j,n'_i) \mid F'(e_j,t_j,n'_i) = (f,z,n_i)\}$$

De même, pour un nœud n'_i d'un graphe de scène de moteur, le SGA renvoie $\emptyset$ ou un nœud d'un graphe de scène de format. F' est la fonction de récupération :

$$F'(e,t,n'_i) = \emptyset \text{ ou } (f,z,n_i)$$

Par conséquent, un nœud d'un graphe de scène de format correspond à aucun, un ou plusieurs nœuds de graphes de scène de moteur d'un environnement virtuel. Ce nœud peut par exemple ne pas avoir été adapté (le nœud racine par exemple), il peut par contre avoir été adapté mais pas pour tous les moteurs (un nœud qui définit une "skybox" n'aura pas d'adaptation dans un graphe de scène de moteur physique). A l'inverse, un nœud d'un graphe de scène de moteur peut correspondre à aucun ou un nœud d'un graphe de scène de format. En effet, certains nœuds d'un graphe de scène de moteur ne correspondent à aucun nœud d'un fichier d'entrée comme par exemple le nœud racine d'un graphe de scène de moteur.

La figure 4.2 illustre ce formalisme. Le SGA s'intègre donc dans un environnement virtuel 3D de façon à réaliser cette tâche dès que l'exécution le requiert (au démarrage et à chaque fois qu'un nœud est ajouté ou modifié). Nous présentons dans le chapitre suivant l'architecture détaillée du SGA ainsi que son fonctionnement.

Chapitre 5

L'ADAPTATEUR DE GRAPHES DE SCÈNE

CETTE partie présente de manière détaillée l'architecture du SGA. Il s'agit d'une architecture modulaire dont l'intégration avec une application d'environnement virtuel 3D est aisée et qui tient compte de la réutilisabilité de ses composants. Elle fournit également une interface de programmation qui peut s'intégrer à n'importe quel environnement virtuel 3D afin de le rendre interopérable avec tous les formats tridimensionnels basés graphe de scène ainsi que tous les composants de rendu réutilisables. La finalité du SGA est de permettre la communication entre un graphe de scène issu d'un fichier en entrée et un graphe de scène d'un composant de rendu en sortie. Nous présenterons tout d'abord la globalité de l'architecture puis nous détaillerons ensuite ses composants, leurs rôles et leurs fonctionnements. Nous verrons ensuite comment le SGA s'intègre dans une application d'environnement virtuel 3D et comment sont gérées les tâches qui lui incombent. Nous conclurons ce chapitre par une discussion sur les propriétés du SGA et de sa conformité avec le cadre fixé.

5.1 PRÉSENTATION DE L'ARCHITECTURE GÉNÉRALE

De par son rôle, le SGA vient se placer entre les fichiers d'entrée de l'application qui génère l'environnement virtuel 3D et les moteurs utilisés par cette dernière.

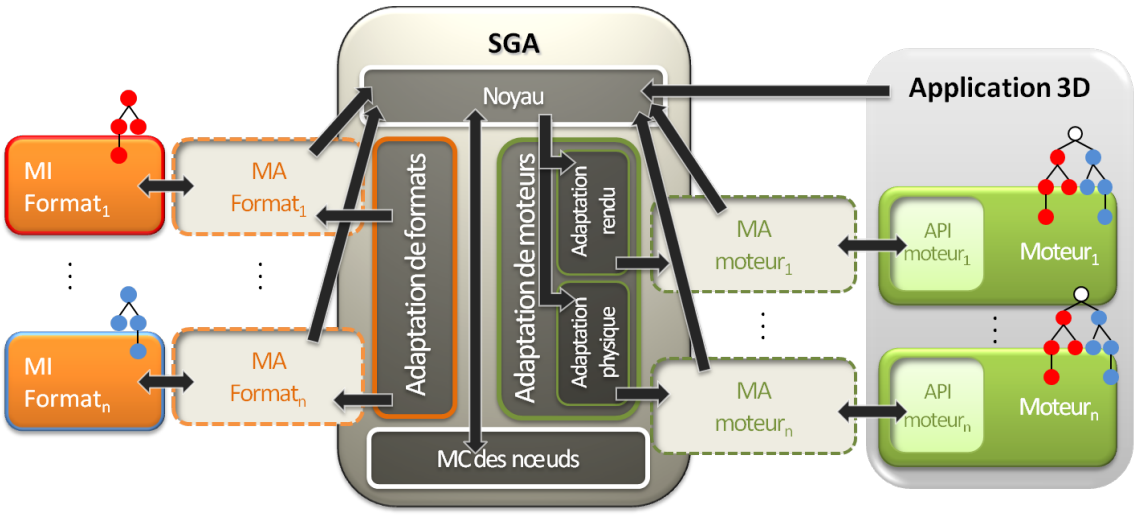


FIGURE 5.1 – *L'architecture du SGA : (MD : module d'interprétation, MA : module d'adaptation et MC : Module de mise en Correspondance)*

En entrée de notre système, nous trouvons les formats 3D et leurs modules d'interprétation dédiés. Le rôle du module d'interprétation est non seulement d'assurer le décodage du fichier mais aussi d'interpréter les fonctionnalités plus avancées du format durant toute la durée de l'application. Il n'est donc pas un simple analyseur syntaxique car son rôle consiste non seulement à décoder le fichier en l'analysant syntaxiquement mais aussi à le décompresser si celui-ci utilise un schéma d'encodage spécifique pour la compression de données. Il peut également contenir un moteur d'exécution dont le rôle consiste quant à lui à exécuter des fonctionnalités dynamiques propres à certains formats, comme la gestion d'événements entre différents nœuds du graphe de scène, l'interpolation de valeurs numériques, ... Notons qu'il n'est pas nécessaire d'implémenter un décodeur de toutes pièces étant donné que la plupart des communautés ou consortiums, spécifiant les différents formats 3D, proposent un module de décodage de référence conforme à la norme. C'est donc le module d'interprétation qui maintient pendant toute l'exécution de l'application, le graphe de scène de format.

A l'autre extrémité du système, en sortie, nous avons l'application qui génère l'environnement virtuel 3D dont le rendu se base sur plusieurs composants dont le nombre et la nature varient suivant les besoins de l'application. L'application peut en effet utiliser un ou plusieurs moteurs de rendu graphique ainsi qu'un moteur physique par exemple. Chacun de ces composants dispose d'une interface de programmation propre fournie par l'éditeur du composant. Celle-ci permet d'accéder et de modifier le graphe de scène de moteur, maintenu au cours de l'exécution de l'application par le moteur.

Le SGA se situe entre ces deux éléments car son rôle consiste à permettre aux graphes de scène de format et aux graphes de scène de moteur d'échanger des informations dans le but de maintenir leur cohérence au cours de l'exécution de l'application. Il doit donc synchroniser l'ensemble des graphes de scène au cours de l'application. Il sera ainsi sollicité au chargement d'un nouveau fichier mais aussi à chaque évènement modifiant l'un des graphes de scène. Ces évènements peuvent être déclenchés aussi bien du côté des formats que du côté des moteurs.

Les derniers éléments de cette architecture sont les modules d'adaptation. Ils sont de deux types : les *modules d'adaptation de format* et les *modules d'adaptation de moteur*. Les modules d'adaptation de format se chargent de l'adaptation du graphe de scène de format géré par leur module d'interprétation associé. De même les modules d'adaptation de moteur se chargent de l'adaptation des graphes de scène générés par leur moteur associé. Les modules d'adaptation sont les seuls éléments à créer pour intégrer la solution du SGA à une application mais ils sont réutilisables d'une application à une autre. La section suivante va détailler chaque composant et expliquer comment ces différents éléments travaillent ensemble.

5.2 LES COMPOSANTS DU SGA

Le SGA est une architecture modulaire, chaque module est en charge d'un format ou d'un moteur. Les interfaces de programmation du SGA se chargent ensuite de centraliser les échanges de données entre ces différents modules.

5.2.1 LE SGA

Le SGA est lui-même constitué de quatre composants : le noyau, une interface de programmation d'adaptation de format, une interface de programmation d'adaptation de moteur et un module de mise en correspondance des nœuds. Ces quatre composants travaillent conjointement pour permettre les échanges de données entre les différents graphes de scène chargés au cours de l'exécution de l'application d'un environnement virtuel 3D.

5.2.1.1 Les interfaces de programmation d'adaptation

Le SGA possède deux interfaces de programmation d'adaptation. La première, l'interface de programmation d'adaptation de formats, est dédiée aux formats 3D. Elle présente plusieurs méthodes qui permettent de modifier et de consulter un graphe de scène de format. Ces méthodes sont appelées par le noyau du SGA lorsqu'un moteur ou l'application demande à modifier ou à consulter un graphe de scène de format. Ces mêmes méthodes sont implémentées ensuite dans les modules d'adaptation de format en utilisant les méthodes mises à disposition par le module d'interprétation du format. Ainsi chaque format implémente les méthodes de l'interface de programmation d'adaptation de format en respectant les règles et le formalisme propres au format. Le tableau 5.1 donne l'ensemble des méthodes disponibles dans l'interface de programmation d'adaptation de format. On peut constater qu'il y a peu de méthodes dédiées à l'interaction, nous n'avons en effet prévu que quelques méthodes basiques car cette fonctionnalité n'est présente que dans peu de formats et est peu évoluée.

Fonction	**Exemples**
Chargement de graphe de scène	*adaptFormat*, *adaptSceneGraph*
Évènements de trames	*updateElapsedTime*
Évènements utilisateur	*onPress*, *onDrag*, *onExit*, *onRelease*, *onEnter*
Consultation	*getNodeFieldValue*
Modification	*setNodeFieldValue*, *matchNodeFieldValue*, *updateNodePosition*, *setViewpoint*

TABLE *5.1* – *Les méthodes de l'interface de programmation d'adaptation de formats.*

La seconde interface de programmation d'adaptation est consacrée aux différents moteurs utilisés pour le rendu de l'environnement virtuel. Elle propose quelques méthodes génériques pour initialiser un graphe de scène de moteur. Ces méthodes sont appelées par le noyau du SGA au démarrage de l'application et lorsque qu'un nouveau graphe de scène de format est chargé. Elles sont par ailleurs implémentées dans chaque module d'adaptation de moteur à l'aide des méthodes fournies par l'interface de programmation du moteur ce qui permet de préserver les particularités propres à chaque moteur. Les différents types de moteurs répondent à des usages très variés, c'est pourquoi nous avons la possibilité de créer pour chaque type une interface de programmation spécialisée qui hérite de l'interface de programmation d'adaptation de moteur générique. Nous avons par exemple une interface de programmation d'adaptation dérivée pour les moteurs de rendu graphique et une autre pour les moteurs physiques. Les méthodes offertes par ces interfaces de programmation dérivées servent à consulter et à modifier les différents graphes de scène de moteurs. Le tableau 5.2 et le tableau 5.3 donnent l'ensemble de ces méthodes pour respectivement les moteurs de rendu et les moteurs physiques.

Il est possible de créer autant d'interfaces de programmation dérivées que nécessaire afin de couvrir tous les types de moteurs utiles.

5.2.1.2 Le noyau

Le noyau est le point d'entrée du SGA ; il centralise tous les appels initiés par l'application, les modules d'interprétation de format ou les moteurs. Son rôle est de synchroniser l'ensemble des graphes de scène (de format et de moteur) qui sont chargés ou générés durant le déroulement de l'application. Lorsqu'un module d'interprétation de format modifie l'un des nœuds de son graphe de format, le noyau va propager cette modification à tous les graphes de moteur concernés qui sont utilisés par l'architecture. L'information est ainsi "poussée" vers la partie rendu de l'application. Dans l'autre sens, lorsqu'un moteur modifie un nœud de son graphe de scène de moteur, le noyau se charge d'en informer le décodeur de format concerné afin que la modification soit effectuée dans le graphe de scène de format. Pour terminer, le noyau propage cette modifica-

Fonction	Exemples
Paramétrage de l'environnement	*createViewpoint, setViewport, createGround*
Paramétrage de caméra	*createCamera, setCamera*
Paramétrage de la navigation	*setNavigation*
Paramétrage de l'éclairage	*createPointLight, createDirectionalLight, createSpotLight*
Paramétrage du graphe de scène	*deleteNode, hideNode, closeNode, showNode, deleteNodeChildren, createGroupNode, attachShapeToSceneGraph*
Paramétrage des géométries	*createSphere, createBox, createMesh, createCylinder, createPlane*
Paramétrage des matériaux	*createMaterial, setMaterial*
Paramétrage de position	*createTransformNode, setTransformNode*
Paramétrage de texture	*createTexture, setTexture*
Consultation	*getNodeTransform, getCurrentTransform*

TABLE *5.2* – *Les méthodes de l'interface de programmation d'adaptation de moteur de rendu.*

Fonction	Exemples
Paramétrage de l'environnement	*setGround, setGravity, setTimeStep*
Paramétrage des géométries	*createBox, createSphere, createCylinder, createMesh, setTransform*
Paramétrage des propriétés physiques	*setNodeMass, setNodeRestitution, setNodeFriction, setNodePhysicsProperties*

TABLE *5.3* – *Les méthodes de l'interface de programmation d'adaptation de moteur physique.*

tion aux autres graphes de scène de moteur. L'information est dans ce cas poussée vers la partie format (contenu) puis "ramenée" vers la partie moteur (rendu).

Le noyau propose plusieurs méthodes qui vont permettre d'initialiser, construire, consulter et modifier des graphes de scène de format et des graphes de scène de moteur. Ces méthodes sont pour la plupart identiques aux méthodes des interfaces de programmation d'adaptation présentées dans le paragraphe 5.2.1.1. La nuance pour les méthodes qui s'adressent à des graphes de scène de format est que leur signature comprend en plus un identifiant de nœud de graphe de scène de moteur, c'est le noyau qui va se charger de retrouver le graphe de scène de format concerné. De la même façon, la signature des méthodes qui s'adressent aux graphes de scène de moteur comporte des identifiants de nœud de graphes de scène de format, le noyau va ensuite propager l'appel à tous les graphes de scène de moteur. Le noyau est aidé dans cette tâche d'association d'identifiants de nœuds par le module de mise en correspondance des nœuds.

5.2.1.3 Le module de mise en correspondance des nœuds

Le module de mise en correspondance des nœuds gère une table de correspondance qui permet pour tous nœuds d'un graphe de scène donné de retrouver le ou les nœuds correspondants dans les autres graphes de scène. Un nœud d'un graphe de scène de format peut avoir un ou plusieurs nœuds qui lui correspondent dans les différents graphes de moteur gérés par l'application. Il peut par exemple avoir un représentant dans le graphe de scène du moteur de rendu et un représentant dans le graphe de scène du moteur physique. A l'inverse, un nœud d'un graphe de scène de moteur a au plus un représentant dans un graphe de scène de format. La figure 5.2 illustre la fonction de l'index de nœuds. Nous avons deux nœuds i issus de deux fichiers différents. Le premier est issu d'un fichier F_1 au format f_1 et a une représentation dans les deux graphes de scène de moteur gérés par l'application. Le second est issu d'un fichier F_2 au format f_2 et a une seule représentation dans graphe de scène du moteur E_1 de type t_1. Le module de mise en correspondance des nœuds est mis à jour au cours du déroulement de l'application, le contenu

de sa table de correspondance évolue en fonction du contexte de rendu (temps écoulé, position de la caméra, ...).

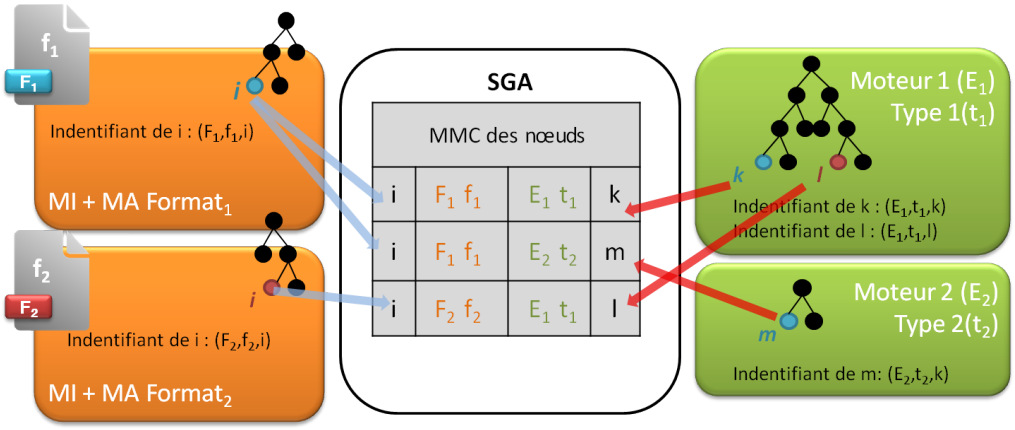


FIGURE 5.2 – *Le fonctionnement du module de mise en correspondance des nœuds (MMC : module de mise en correspondance, MI : module d'interprétation et MA : module d'adaptation) : le référencement de deux nœuds i issus de deux fichiers différents.*

Le bon fonctionnement du module de mise en correspondance des nœuds repose sur l'identification des nœuds indexés. Cette identification doit être impérativement unique, aussi le SGA fournit une méthode de nommage adéquat qui garantit l'unicité de cet identifiant. A chaque création d'un nouveau nœud, cette méthode est appelée afin de créer un identifiant unique pour ce nœud dans son graphe de format d'origine mais aussi dans tous les graphes de moteur où une instance de ce nœud sera créée. L'identifiant d'un nœud d'un graphe de scène de format comprend le nom de son fichier d'origine, le format du fichier ainsi qu'un identifiant du nœud dans son graphe de format. Ce dernier identifiant peut être fourni dans le fichier via une notation de nommage propre au format ou, s'il n'existe pas, être généré par le SGA. Dans le cas d'un nœud d'un graphe de scène de moteur, le caractère d'unicité de son identifiant est garanti par le nom du moteur, son type et un identifiant du nœud dans le graphe de moteur. Ce dernier est généré par le SGA au moment de la création de ce nœud. La mise à jour du module de mise en correspondance des nœuds est assurée par le noyau du SGA.

5.2.2 LES MODULES D'ADAPTATION

Les modules d'adaptation sont les composants qui vont permettre d'un côté la communication entre les modules d'interprétation de formats et le SGA et aussi, de l'autre côté, entre les moteurs et le SGA. Pour assurer cette communication dans les deux sens, leur rôle est décomposé en deux tâches que nous qualifions d'adaptation et d'adapté. Ces composants sont dédiés à un format ou à un moteur et sont réutilisables d'une application à une autre.

5.2.2.1 Les modules d'adaptation de formats

Comme leur nom l'indique, les modules d'adaptation de formats sont dédiés à un format, il y en a un pour chaque format pris en charge. Les modules d'adaptation de format ne dépendent pas d'une application, ils sont réutilisables ; cependant, ils dépendent d'un module d'interprétation. Ils reposent en effet sur la représentation du graphe de scène utilisée par le module d'interprétation pour procéder à son adaptation. De plus, selon les fonctionnalités prises en charge par le module d'interprétation, le module d'adaptation de format peut se charger de plus ou moins des propriétés définies par le format. Il peut ainsi pallier les manques et les incomplétudes du module d'interprétation utilisé. Il est également possible d'avoir plusieurs versions d'un module

d'adaptation de format en fonction du décodeur sur lequel il se base, d'un profil donné d'une norme à laquelle est rattaché le format mais aussi en fonction du mode d'adaptation choisi pour certaines fonctionnalités. Dans le cas d'un nœud de niveau de détail par exemple, il est possible de calculer la sélection du niveau de détail par rapport à un point de vue donné par le module d'interprétation si la fonctionnalité est disponible, ou par le module d'adaptation si elle a été implémentée. Dans ce cas, seuls les noeuds couramment sélectionnés seront transmis aux graphes de moteurs ; le module d'interprétation se chargera de modifier celui-ci lorsque la position de la caméra le requerra. Une autre adaptation possible consiste à transférer l'ensemble des niveaux de détail lorsque les moteurs utilisés proposent une fonctionnalité de sélection de niveau de détail intégrée et souvent optimisée.

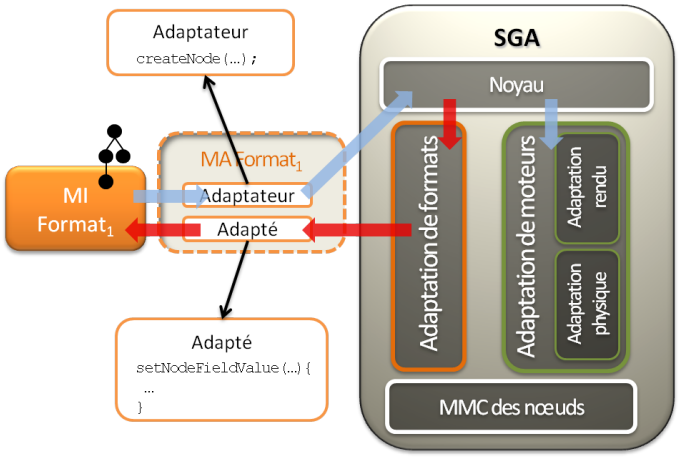


FIGURE *5.3 – Les modules d'adaptation de format : les flèches bleues montrent le chemin de communication du module d'interprétation de format vers le noyau du SGA et les flèches rouges celui du noyau du SGA vers le module d'interprétation de format.*

Le rôle du module d'adaptation de format se divise en deux tâches : une tâche d'**adaptateur** et une tâche d'**adapté**. La tâche d'*adaptateur* consiste à adapter le graphe de scène de format aux différents graphes de scène de moteur. Pour cela, le module d'adaptation utilise les méthodes proposées par le noyau. Ainsi, lorsqu'un nouveau nœud est atteint par l'analyseur syntaxique du module d'interprétation, le module d'adaptation de format va appeler des méthodes du noyau pour créer des instances de ce nœud dans tous les graphes de moteur concernés. La tâche d'*adapté* consiste quant à elle à adapter les modifications des graphes de scène de moteur dans le graphe de scène de format. Ces modifications transitent par le noyau du SGA qui les traite par des appels aux méthodes de l'interface de programmation d'adaptation de format. Celles-ci sont implémentées dans le module d'adaptation de format en utilisant les méthodes fournies par le décodeur. De cette façon, ces instructions sont adaptées aux spécifications du format. Néanmoins, l'interface de programmation d'adaptation de format étant généraliste, certaines méthodes la composant n'ont pas d'équivalent dans certains formats. Aussi le module d'adaptation de format n'implémente t-il pas nécessairement toutes ces méthodes. La figure 5.3 illustre les deux tâches des modules d'adaptation de format ainsi que leur intégration avec les modules d'interprétation de format et le SGA. Les requêtes qui relèvent de la tâche d'*adaptateur* du module d'adaptation transitent du module d'interprétation vers le module d'adaptation puis sont gérées par le noyau du SGA. À l'inverse, les requêtes qui relèvent de la tâche d'*adapté* transitent du noyau vers le module d'adaptation de format pour être ensuite traitées par le décodeur de format. Si cette requête induit une modification du graphe de scène de format alors elle donnera lieu à une nouvelle requête du ressort de l'adaptateur du module d'adaptation et sera ainsi propagée à l'ensemble des graphes de moteurs.

5.2.2.2 Les modules d'adaptation de moteur

Les modules d'adaptation de moteur sont eux dédiés à un moteur particulier et comme les modules d'adaptation de format, ils ne dépendent pas d'une application. Ils sont donc également réutilisables d'une application à une autre. Ils reposent sur l'interface de programmation mise à disposition par le moteur qu'ils adaptent.

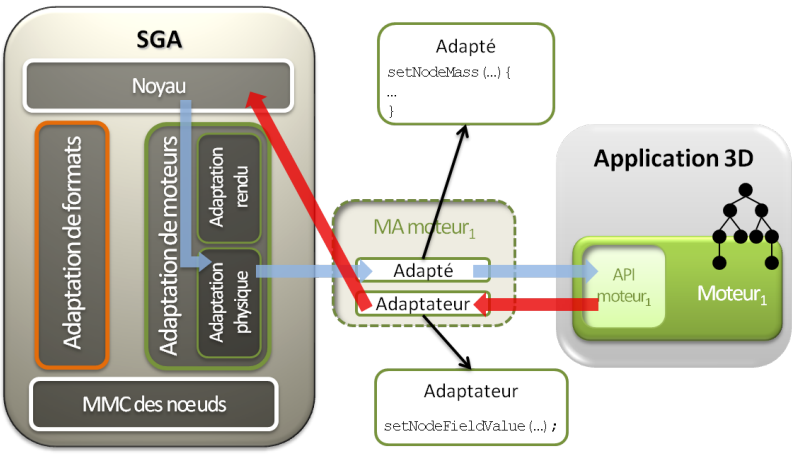


FIGURE 5.4 – *Les modules d'adaptation de moteur : les flèches rouges montrent le chemin de communication du moteur vers le noyau du SGA et les flèches bleues celui du noyau du SGA vers le moteur.*

Comme le montre la figure 5.4, leur rôle se divise également en une tâche d'**adaptateur** et une tâche d'**adapté**. La tâche d'*adaptateur* consiste à adapter un nœud du graphe de scène de moteur au nœud correspondant dans son graphe de scène de format. Pour cela, le module d'adaptation va appeler les différentes méthodes proposées par le noyau. La tâche d'*adapté* consiste quant à elle à implémenter les méthodes de l'interface de programmation d'adaptation de moteurs ainsi que celles de l'interface de programmation d'adaptation dérivée qui est dédiée au type du moteur. Ainsi, pour un moteur physique, il faudra implémenter les méthodes de l'interface de programmation d'adaptation de moteurs ainsi que celles de l'interface de programmation d'adaptation de moteurs physiques. Si certaines de ces méthodes correspondent à des fonctionnalités qui ne sont pas offertes par le moteur, elles ne seront pas implémentées.

5.3 L'INTÉGRATION DU SGA DANS UN ENVIRONNEMENT VIRTUEL 3D

Nous avons vu dans le paragraphe précédent le rôle de chaque composant, la façon dont les composants fonctionnent ainsi que la façon dont ils interagissent. Nous allons maintenant voir comment cette architecture peut être intégrée à un environnement virtuel 3D et comment sont traitées les principales actions.

5.3.1 LA PHASE DE DÉMARRAGE ET D'INITIALISATION

L'environnement 3D est simulé par une application 3D. Cette application détermine les différents moteurs sur lesquels elle repose ainsi que les fichiers 3D qu'elle va charger. C'est donc à son initiative que les moteurs seront initialisés et que les fichiers seront chargés. La figure 5.5 montre les séquences échangées entre les différents composants de l'architecture au démarrage de l'application. Tout d'abord l'application référence tous les modules d'adaptation de format qu'elle a à disposition sur sa machine d'exécution (message 1). Pour chaque module trouvé, elle en informe le noyau du SGA qui met à jour au fur et à mesure une table qui indique, pour chaque extension de fichier prise en charge, le module d'adaptation correspondant (messsage 2). Ensuite, on initialise

le rendu de l'application (message 3) en chargeant l'ensemble des modules d'adaptation de moteur nécessaires au fonctionnement de l'application. Chaque module d'adaptation de moteur est ainsi chargé (message 4), ce qui met à jour une table de tous les modules d'adaptation de moteur utilisés par l'application. Enfin, l'application demande leur initialisation (message 5) au noyau. Le noyau appelle alors la méthode d'initialisation prévue par l'interface de programmation d'adaptation de moteur (message 6) dont l'implémentation repose sur l'utilisation de méthodes propres à l'interface de programmation de chaque moteur (message 7).

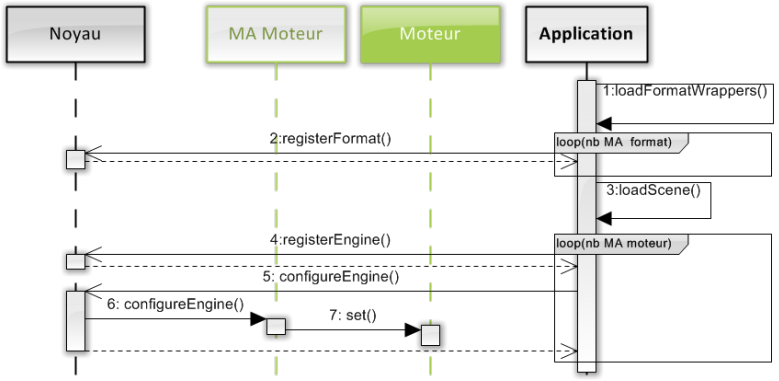


FIGURE *5.5 – Exemple de messages échangés par les composants du SGA lors du démarrage de l'application.*

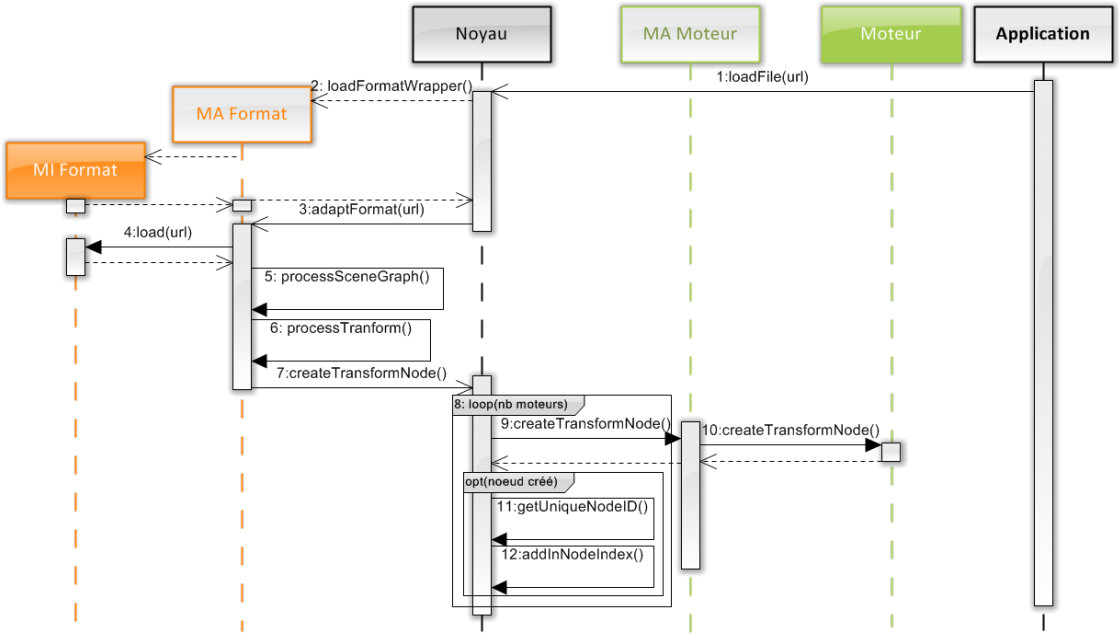


FIGURE *5.6 – Exemple de messages échangés par les composants du SGA lors du chargement d'un fichier 3D.*

Après cette phase d'initialisation du SGA, l'application va charger les différents fichiers qui vont composer la scène rendue. La figure 5.6 montre les messages échangés lors du chargement d'un fichier. L'application commence par transmettre au noyau l'adresse du fichier à charger (message 1). Le noyau va alors charger le module d'adaptation de format approprié (message 2) puis

demander l'adaptation du fichier à celui-ci (message 3). Le module d'adaptation de format utilise alors les méthodes fournies par le décodeur de format pour obtenir le graphe de scène encodé dans le fichier (message 4). Il commence alors l'adaptation du graphe de scène de format en parcourant celui-ci (message 5). A chaque nœud atteint, il appelle la méthode adéquate pour effectuer son adaptation dans les différents graphes de scène de moteur. Pour un nœud de positionnement par exemple (message 6), il va appeler le noyau (message 7). Le noyau va alors parcourir la liste des moteurs utilisés (boucle 8) et pour ceux dont le type implique la prise en compte des nœuds de positionnement, il va demander l'adaptation de ce nœud au module d'adaptation de moteur (message 9). Le module d'adaptation de moteur va ensuite se servir des méthodes fournies par l'interface de programmation du moteur pour adapter ce nœud dans son graphe de scène de moteur (message 10). Si cette opération réussit, le noyau va créer un identifiant unique pour ce nœud de graphe de scène de moteur (message 11) et le référencer dans le module de mise en correspondance des nœuds (message 12).

5.3.2 La phase d'exécution de l'application

Pendant le déroulement de l'application, le SGA continue son rôle de synchronisation des graphes de scène mis en relation par l'application aussi bien du côté des formats que des moteurs. La figure 5.7 donne un exemple des messages échangés entre les modules du SGA pour effectuer cette synchronisation pour les graphes de scène de format. A chaque mise à jour du temps par l'application (boucle 1), celle-ci transmet le temps écoulé au noyau (message 2). Le noyau propage ensuite l'information à tous les modules d'adaptation, ici nous ne nous intéressons qu'aux modules d'adaptation de format (boucle 3). Pour un module d'adaptation de format donné, lorsqu'il reçoit la mise à jour du temps écoulé (message 4), il la transmet à son décodeur (message 5) dans le cas où le format de fichier en question tient compte du temps. Une fois que le décodeur a géré cette mise à jour, si celle-ci a abouti à la modification d'un ou plusieurs nœuds, le module d'adaptation de format va alors en informer les graphes de scène de moteur via le noyau (boucle 6 et message 7). Pour chaque moteur lié à l'application (boucle 8), le noyau va ensuite propager cette modification (message 9) qui sera prise en charge ensuite par le module d'adaptation de moteur (message 10).

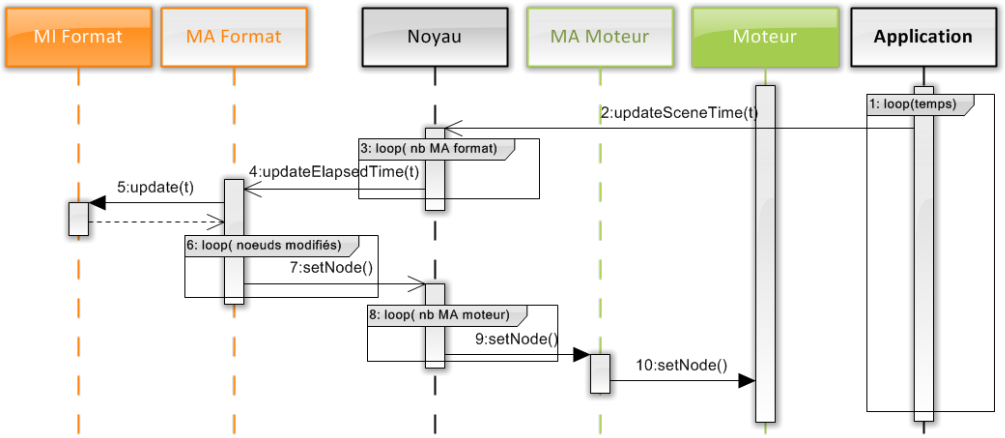


FIGURE 5.7 – *Exemple de séquences échangées lors de la mise à jour du temps écoulé pour les contenus.*

Le noyau informe également les modules d'adaptation de moteur de la mise à jour du temps. Pour cela, il appelle une méthode de l'interface de programmation d'adaptation de moteur dont l'implémentation diffère en fonction des moteurs. La figure 5.8 montre un exemple des messages échangés lors de la mise à jour du temps pour un moteur physique. Le noyau reçoit la mise à

jour du temps par l'intermédiaire de l'application (boucle 1 et message 2). Il en informe ensuite l'ensemble des moteurs liés à l'application (boucle 3) en appelant la méthode dédiée de l'interface de programmation d'adaptation de moteurs (message 4). Le module d'adaptation de moteur propage cette information à son moteur (message 5). Lorsque le type du moteur tient compte du temps, cette mise à jour peut entraîner une modification de son graphe de scène de moteur, le module d'adaptation de moteur en informe le noyau pour chaque nœud modifié (message 6). À l'aide du module de mise en correspondance des nœuds, il retrouve le module d'adaptation de format concerné ainsi que l'identifiant du nœud correspondant dans son graphe de scène de format (message 7). Le noyau peut alors demander la mise à jour du nœud au module d'adaptation de format (message 8) qui propage cette requête au module d'interprétation (message 9). Si cette modification réussit, le module d'adaptation de format la propage ensuite aux autres graphes de scène de moteur par l'intermédiaire du noyau (message 10). Celui-ci se charge d'en informer tous les moteurs (boucle 11 et message 12) qui synchronisent à leur tour leur graphe de scène de moteur (message 13).

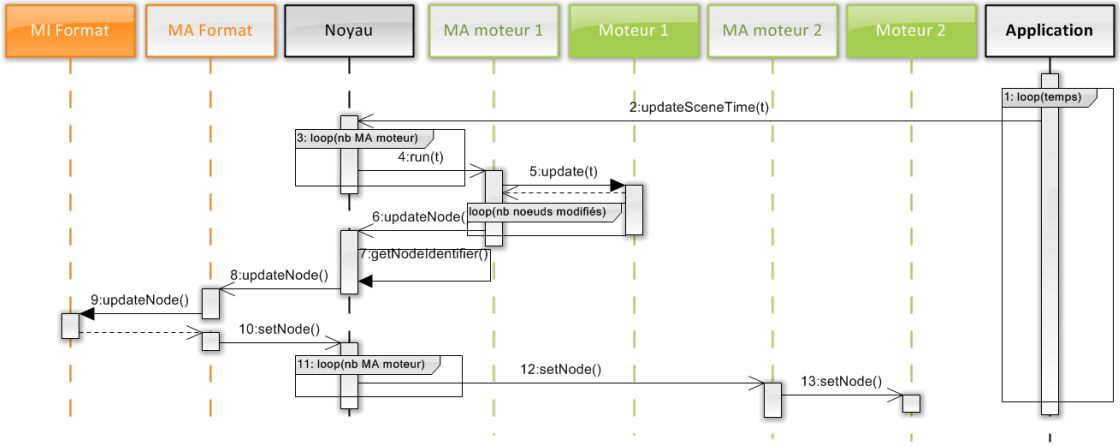


FIGURE *5.8 – Exemple de séquences échangées lors de la mise à jour du temps écoulé pour un moteur physique.*

Nous pouvons voir à travers ces opérations courantes effectuées par le SGA que l'introduction et le chargement d'un fichier d'un nouveau format est une tâche triviale. Il suffit de fournir à l'application un module d'adaptation pour ce format ainsi que son module d'interprétation pour que l'application soit capable de charger n'importe quel fichier encodé dans ce nouveau format. De plus, l'introduction ou la suppression de l'un des moteurs de l'application n'a également que peu d'impact sur celle-ci.

5.4 LE SGA : DISCUSSION ET FONCTIONNALITÉS

Dans cette partie nous allons démontrer que le SGA répond à la problématique à l'origine de ces travaux. Nous allons discuter de la conformité du SGA avec le cadre fixé par la problématique et par l'étude bibliographique.

5.4.1 INTEROPÉRABILITÉ AVEC LES CONTENUS 3D

Le SGA est compatible avec la plupart des contenus 3D, il est en effet compatible avec la quasi-totalité des formats 3D. Les seuls formats à faire exception sont ceux qui ne sont pas basés sur une représentation en graphe de scène comme par exemple le format *.obj*. Cependant, si pour un format de ce type il existe un décodeur capable de construire un graphe de scène à partir des

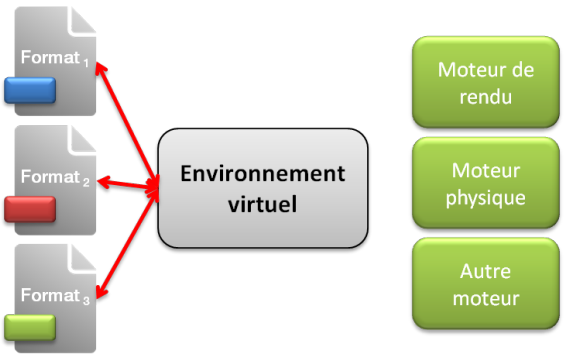


FIGURE 5.9 – *L'interopérabilité avec les contenus 3D.*

données contenues dans ce fichier, alors cela rend possible son utilisation avec le SGA. Le système des modules d'interprétation a également d'autres avantages. Tout d'abord, cela permet de gagner du temps de développement car il est constitué d'outils déjà disponibles. Chaque format tridimensionnel possède tout au moins un analyseur syntaxique or il s'agit du composant de base du module d'interprétation. Si le format intègre en plus d'autres fonctionnalités, les outils permettant de les interpréter sont fournis par l'éditeur du format et pour les formats les plus populaires, un large choix d'outils est généralement disponible. Ce système a également l'avantage d'éviter les erreurs et les pertes d'information dans les contenus. En effet, si les outils utilisés dans le module d'interprétation sont fiables, le graphe de scène de format sur lequel il repose est exempt d'erreurs. Par ailleurs, l'ensemble des fonctionnalités propres au format et les mécanismes qui les mettent en œuvre sont entièrement pris en charge par le module d'interprétation ce qui assure leur conformité avec les spécifications du format. Le test de conformité de l'adaptation est déportée sur le module d'interprétation. L'adaptation opérée par le SGA n'effectue aucune interprétation sur le contenu contrairement à ce que fait une conversion. L'adaptation recopie l'état d'un nœud à un instant donné, cet état étant duplicable du fait des similitudes dans les représentations utilisées par les composants de rendu.

Notre architecture permet l'interopérabilité des contenus avec les environnements virtuels 3D. A condition d'avoir le module d'adaptation de format approprié, grâce au SGA, nous pouvons utiliser n'importe quel format dans un environnement virtuel 3D sans être gêné par les composants de rendu qui sont utilisés. Le choix d'un composant de rendu, notamment d'un moteur de rendu, pour un environnement virtuel 3D n'est alors plus synonyme de restriction en terme de formats compatibles. De plus, cette compatibilité ne se fait pas en contrepartie d'une perte d'intégrité des informations encodées dans le fichier.

Nous pouvons également facilement mixer plusieurs fichiers issus de formats différents dans une même scène et ainsi mixer leurs fonctionnalités. Nous pouvons par exemple utiliser un premier fichier pour décrire un monde et l'utiliser pour explorer des modèles encodés dans des formats qui ne permettent pas de décrire le monde qui entoure le modèle.

5.4.2 INTEROPÉRABILITÉ AVEC LES COMPOSANTS DE RENDU

Grâce aux modules d'adaptation de moteur, qui permettent d'interfacer n'importe quel composant de rendu avec le SGA, ce dernier est compatible avec tous les environnements virtuels 3D quels que soient les composants de rendu qu'ils utilisent. Leur implémentation est relativement aisée, il s'agit essentiellement d'implémenter les fonctions qui relèvent de la tâche d'adapté de l'interface de programmation d'adaptation de moteur en appelant les méthodes de l'interface de programmation du moteur. Il permet également une grande souplesse d'utilisation de ces composants car ceux-ci peuvent être aisément intervertis, ajoutés ou supprimés en fonction des besoins de l'application. Le SGA permet également aux différents moteurs utilisés par l'application de communiquer grâce à la tâche d'adaptation des modules d'adaptation. Son implémentation

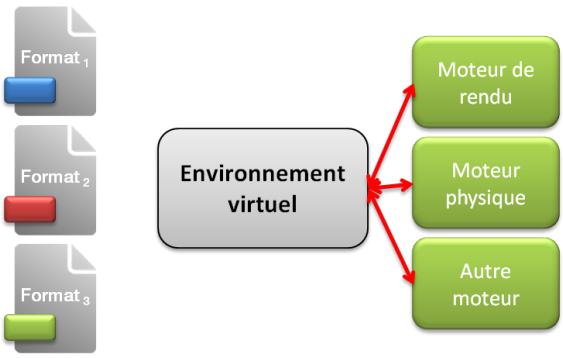


FIGURE *5.10 – L'interopérabilité avec les composants de rendu.*

consiste à faire des appels aux méthodes appropriées du noyau du SGA. De cette façon, les différents rendus générés par chacun des moteurs peuvent être exploités par les autres moteurs.

5.4.3 INTEROPÉRABILITÉ ENTRE LES CONTENUS ET LES COMPOSANTS DE RENDU

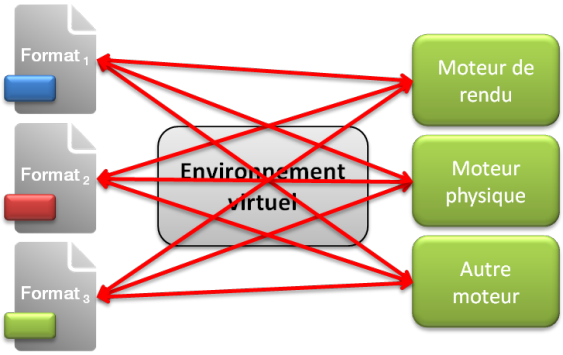


FIGURE *5.11 – L'interopérabilité entre les contenus et les composants de rendu.*

L'utilisation du SGA permet les interactions entre les composants de rendu de l'environnement virtuel 3D et ses contenus. Les modifications d'un graphe de scène sont propagées aux autres graphes de scène comme expliqué dans la partie 5.3. Le SGA permet également un premier niveau d'interaction entre les fichiers. Un second niveau d'interaction est possible et sera décrit dans le chapitre 7. Ce premier niveau d'interaction autorise la création de scènes composées de contenus issus de fichiers de différents formats. Au sein de la scène ainsi constituée, les objets interagissent car ils sont soumis aux règles du même monde (gravité, comportement, ...). Deux objets issus de formats différents peuvent ainsi entrer en collision par exemple.

Le SGA offre deux façons de créer une scène composée de contenus encodés dans des formats différents. La première solution est de les charger tour à tour séparément. Chaque contenu est encodé dans un fichier et chaque fichier est pris en pris en charge par un couple "module d'interprétation-module d'adaptation" adéquat. Dans ce cas de figure, les graphes de scène de format seront adaptés dans les graphes de scène de moteur comme des enfants du nœud racine de ce dernier (cf schéma 5.12).

Cette façon de faire ne permet pas de placer les modèles les uns par rapport aux autres mais permet de les visualiser simultanément dans le même monde.

La seconde solution consiste à utiliser le mécanisme de référencement de fichier proposé par les formats. En effet la plupart des formats 3D offrent la possibilité de référencer un fichier externe au sein d'un autre fichier du même format tel le nœud "inline" de VRML qui permet d'inclure un

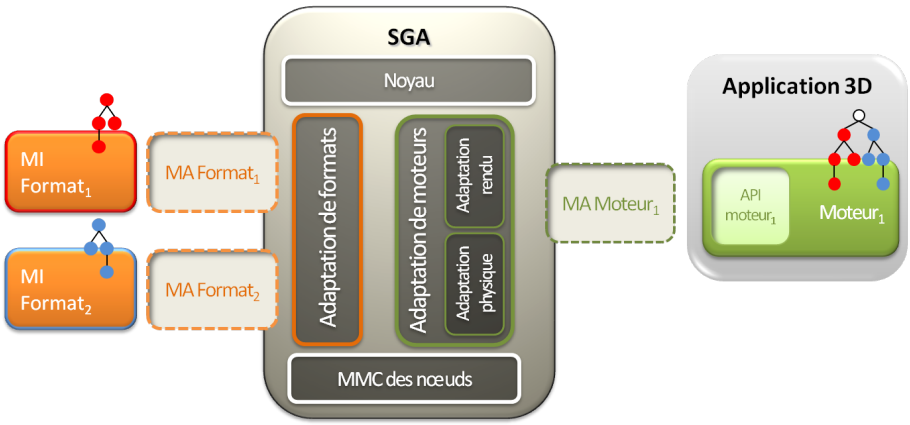


FIGURE 5.12 – *Composition de scène avec le SGA grâce au chargement séparé : le graphe de scène rouge du format 1 et le graphe de scène bleu du format 2 sont tous les deux fils de la racine du graphe de scène de moteur.*

graphe de scène encodé dans un autre fichier. Grâce au SGA, nous avons la possibilité d'utiliser cette fonctionnalité mais pas seulement en référençant un fichier du même format. Nous pouvons ainsi inclure un sous-graphe de scène encodé dans un fichier de format F_2 dans un graphe de scène encodé dans un fichier de format F_1 comme l'illustre la figure 5.13.

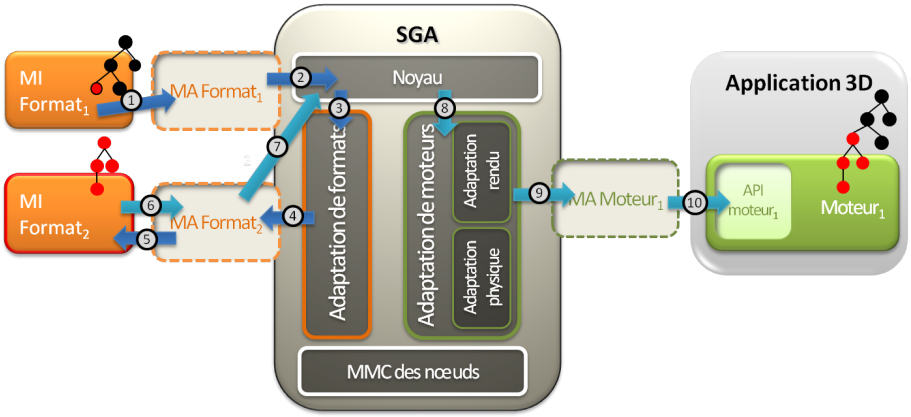


FIGURE 5.13 – *Composition de scène avec le SGA grâce au chargement avec un fichier conteneur : le graphe de scène rouge du format 2 est placé, dans le graphe de scène de moteur, au niveau de sa référence dans le graphe de scène noir du format conteneur.*

Cette situation est facilement gérée par le SGA. Au moment du parcours du graphe de scène principal, lorsque qu'un nœud de référencement est atteint (étape 1), le module d'adaptation de format appelle le noyau (étape 2). Le noyau charge alors le module d'adaptation approprié en fonction de l'extension du fichier référencé (étapes 3 et 4). Le module d'adaptation appelle ensuite le module d'interprétation (étape 5) qui va alors charger le graphe de scène encodé dans le fichier (étape 6) puis commencer son adaptation en graphes de scène de moteur (étapes 7,8,9 et 10). Le diagramme 5.14 montre les messages échangés pour réaliser cette opération.

Cette possibilité a plusieurs avantages :

- il est possible de charger plusieurs fichiers à partir d'un seul fichier,
- cela permet d'inclure des fichiers de manière structurée, pas seulement comme des enfants de l'unique nœud racine. Le nœud de référencement peut en effet être lui-même encapsulé dans un nœud de positionnement de façon à décrire la position et d'ajuster la dimension du

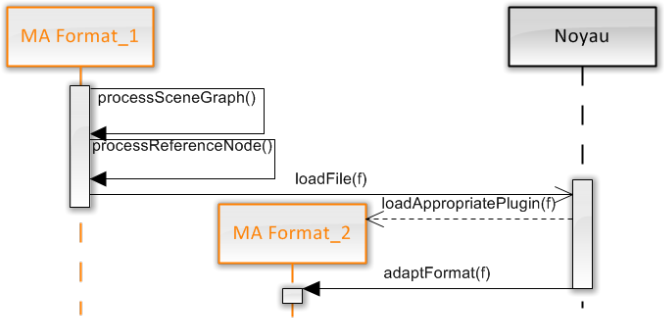


FIGURE *5.14 – Exemple de messages échangés lors du traitement d'un fichier de format F_2 référencé dans un fichier de format F_1.*

modèle ainsi importé.
- cela rend possible l'application d'interactions décrites dans le fichier principal à des contenus qui ne sont pas du même format. Nous pouvons ainsi animer des modèles encodés dans des formats qui n'offrent pas cette fonctionnalité.

5.5 SYNTHÈSE ET CONCLUSION

Le SGA par sa conception modulaire permet de répondre au problème soulevé dans le chapitre 2. Il offre la possibilité de rendre interopérables tous les formats 3D basés graphe de scène avec n'importe quel environnement virtuel 3D. Il répond également aux caractéristiques que nous avions mises en évidence dans l'étude de l'état de l'art. Le tableau 7.4.2 résume ce cadre ainsi que ces caractéristiques et indique si le SGA les satisfait.

	Caractéristiques de la solution	**SGA**
Cadre initial	utilisation directe des contenus	✓
	utilisation directe des composants logiciels	✓
	communication entre contenus et composants	✓
	communication inter-contenus	✓
	communication inter-composants	✓
Contenus	préserve les fonctionnalités du format des fichiers chargés	✓
	pas de conversion	✓
	préserve les modèles encodés dans les fichiers chargés	✓
Composants de rendu	collaboration possible entre eux	✓
Environnements 3D	pas de modification préalable des contenus	✓
	pas de mise en place de protocole	✓

TABLE *5.4 – Le SGA et les caractéristiques d'une solution d'interopérabilité satisfaisante.*

Chapitre 6

IMPLÉMENTATION DE L'ADAPTATEUR DE GRAPHES DE SCÈNE ET EXEMPLES D'INSTANCIATIONS DE MODULES D'ADAPTATION

LE chapitre précédent a présenté en détail le SGA et son fonctionnement. Nous allons maintenant expliquer comment nous avons implémenté cette architecture et comment nous avons instancié plusieurs modules d'adaptation que nous avons réalisé. Nous commencerons par détailler l'implémentation en expliquant et justifiant les choix qui ont été faits. Nous présenterons ensuite quelles instanciations nous avons mises en œuvre afin d'évaluer la faisabilité et la conformité du SGA à notre cadre d'étude. Nous montrerons ensuite dans quelle mesure cette application de test répond à notre problématique. Nous conclurons ce chapitre sur une étude critique du SGA et une discussion sur les améliorations possibles.

6.1 NOTRE IMPLÉMENTATION DU SGA

Pour mettre en œuvre le SGA, notre choix s'est porté sur le langage C++ car il possède beaucoup d'avantages pour une application d'informatique graphique (rapidité, performance, libre de droit, ...). De plus, la grande majorité des interfaces de programmation proposées par les moteurs sont développées en C++, le choix de ce langage facilite donc leurs intégrations avec le SGA. Notre implémentation se découpe en quatre classes principales : le noyau (`SGAKernel`), une fabrique de modules d'adaptation de format (`FormatWrapperFactory`), une fabrique de modules d'adaptation de moteur (`EngineWrapperFactory`) et un index des nœuds (`NodeIndexer`).

6.1.1 LE NOYAU

Au centre du SGA, nous avons le noyau (`SGAKernel`), point d'entrée de notre architecture pour les modules d'adaptation et l'application. Pour remplir son rôle, le SGA expose plusieurs méthodes dans son interface de programmation et dispose de plusieurs structures de données (cf 6.1) :

- une table des modules d'adaptation de format disponibles : `m_AvailableFormatWrapperMap`,
- une table des modules d'adaptation de format chargés (utilisée pour le chargement dynamique des modules d'adaptation de format) : `m_LoadedWrapper`,
- un pointeur vers la fabrique des instances de modules d'adaptation de format (une instance par fichier chargé par l'application) : `m_EngineWrapper`,

– un pointeur vers la fabrique des instances de modules d'adaptation de moteur (une instance par moteur utilisé par l'application) : `m_FormatWrapper`.

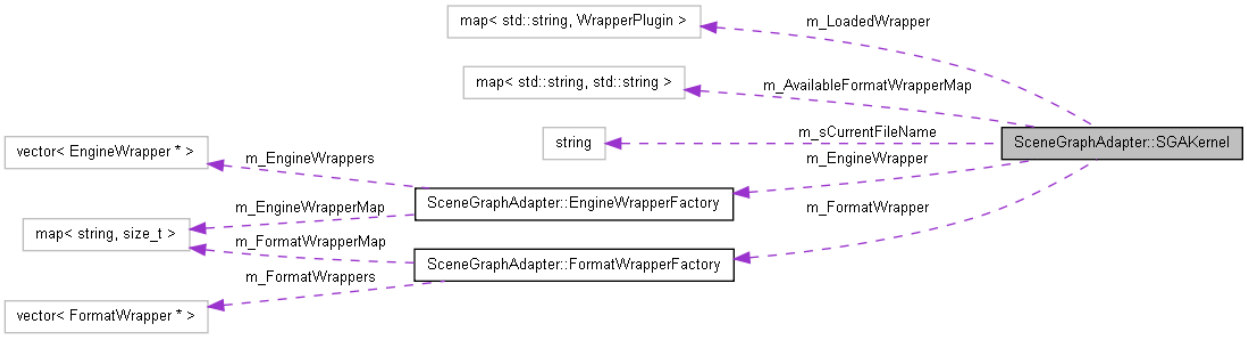


FIGURE *6.1 – La classe du noyau du SGA et ses attributs.*

6.1.2 LA FABRIQUE DES INSTANCES DE MODULES D'ADAPTATION DE FORMAT

La fabrique des modules d'adaptation de format (cf figure 6.2) gère l'ensemble des modules d'adaptation de format utilisés par le SGA au cours du déroulement de l'application. A chaque nouveau fichier chargé par l'application, la fabrique crée une nouvelle instance du module d'adaptation de format (`FormatWrapper`) appropriée pour ce type de fichier. La fabrique référence toutes ces instances et à chacune d'elles, elle associe une configuration (`WrapperConfiguration`). Il s'agit d'une structure de données qui donne des informations sur le module d'adaptation de format : son type (`m_type`) qui sera "format" dans le cas présent, sa version (`m_version`), son adresse (`m_identifier`) et l'extension du format pris en charge (`m_extension`). Le noyau du SGA se sert ensuite de la fabrique de modules d'adaptation de format pour accéder aux différentes instances de modules d'adaptation dont l'application a besoin. Les instances de module d'adaptation de format héritent de l'interface de programmation d'adaptation de format que nous avons définie dans une classe virtuelle appelée `FormatAdapter`.

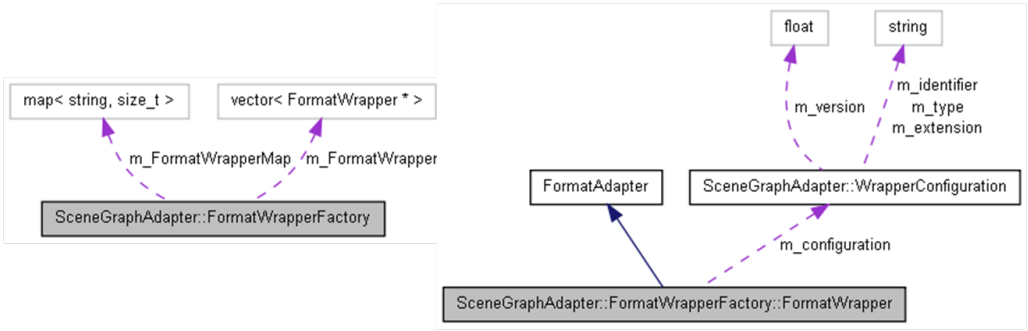


FIGURE *6.2 – La fabrique des instances de modules d'adaptation de format et ses attributs.*

6.1.3 LA FABRIQUE DES INSTANCES DE MODULES D'ADAPTATION DE MOTEUR

La fabrique des instances de modules d'adaptation de moteurs (cf figure 6.3) gère l'ensemble des modules d'adaptation de moteur utilisés par le SGA au cours du déroulement de l'application. C'est elle qui crée une nouvelle instance de module d'adaptation de moteur au démarrage de l'application pour chaque moteur requis par l'application. Ensuite, au cours du déroulement de

l'application, elle permet au noyau du SGA d'atteindre l'ensemble des modules d'adaptation de moteur.

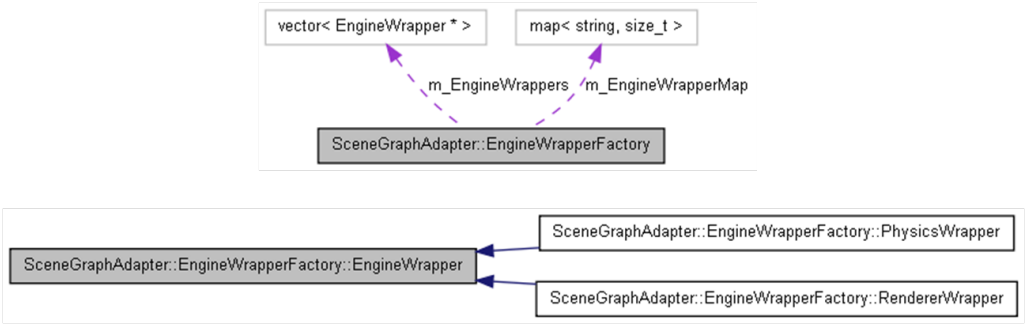


FIGURE 6.3 – *La fabrique des instances de modules d'adaptation de moteur et ses attributs.*

Dans notre implémentation, nous prenons en compte deux types de moteur : les moteurs de rendu et les moteurs physiques. La fabrique des instances de modules d'adaptation de moteur gère donc deux types de module d'adaptation : les modules d'adaptation de moteur de rendu (`RendererWrapper`) et les modules d'adaptation de moteur physique (`PhysicsWrapper`). Les instances de module d'adaptation de moteur de rendu héritent de l'interface de programmation d'adaptation de moteur de rendu que nous avons définie dans une classe virtuelle appelée `RendererAdapter` alors que les instances de module d'adaptation de moteur physique héritent de l'interface de programmation d'adaptation de moteur physique définie dans la classe virtuelle `PhysicsAdapter` (cf figure 6.4).

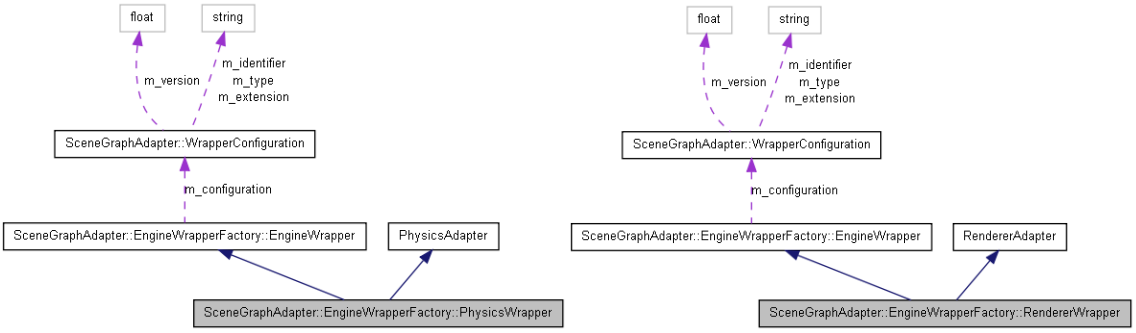


FIGURE 6.4 – *La fabrique des instances de modules d'adaptation de moteur et ses attributs.*

A chaque instance de module d'adaptation de moteur est associée une configuration (`WrapperConfiguration`) qui donne des informations sur ce dernier : son type (`m_type`) qui sera "rendu" ou "physique" selon le cas, sa version (`m_version`) et son identifiant (`m_identifier`) qui est généralement le nom du moteur. L'attribut `m_extension` n'est renseigné que pour les modules d'adaptation de format.

6.1.4 LE MODULE DE MISE EN CORRESPONDANCE DES NŒUDS

Dernier élément essentiel dans l'implémentation du SGA, le module de mise en correspondance des nœuds (cf 6.5) est quant à lui implémenté à l'aide de deux tables : une "map" pour retrouver un nœud d'un graphe de scène de format à partir d'un nœud d'un graphe de scène de moteur (`m_EngineToFormatMap`) et une "multimap" pour retrouver les nœuds correspondants à un nœud d'un graphe de scène de format dans les différents graphes de scène de moteur gérés par l'application (`m_FormatToEngineMap`).

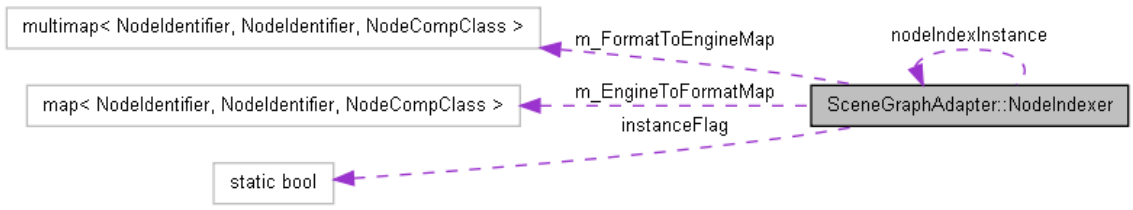


FIGURE *6.5 – L'implémentation de l'index de nœuds.*

6.2 L'INSTANCIATION DU SGA

Nous avons choisi d'instancier notre implémentation du SGA avec deux types de formats et deux moteurs. Côté formats, nous prenons en charge X3D et Collada. Par ailleurs, côté moteurs nous avons choisi un moteur de rendu, Ogre, et un moteur physique, Bullet. Nous allons dans la suite de ce paragraphe justifier ce choix d'instanciation et préciser les détails techniques le concernant.

6.2.1 LES FORMATS

Les formats X3d et Collada sont deux formats très complémentaires de part leurs fonctionnalités, c'est pourquoi nous les avons choisi pour l'évaluation du SGA. Ils partagent néanmoins la propriété d'avoir une syntaxe basé XML mais leurs structurations et leurs organisations respectives sont très différentes.

6.2.1.1 Instanciation du module d'adaptation X3D

FIGURE *6.6 – L'instance de X3D : son module d'interprétation (MI) et son module d'adaptation (MA).*

Le format X3D, en plus d'être libre de droit, présente plusieurs caractéristiques qui nous paraissent essentielles pour l'évaluation du SGA. C'est en effet l'un des formats les plus complet en terme de fonctionnalités et notamment pour l'interactivité de la scène. Nous avions besoin d'un format proposant de nombreuses fonctionnalités afin de s'assurer que le SGA satisfait l'un des critères exigés par notre cadre d'étude : la préservation de l'ensemble des fonctionnalités d'un format. D'autre part, il est bien documenté et beaucoup de fichiers X3D d'exemples sont disponibles. Enfin, c'est un format basé XML ce qui facilite son analyse.

Le module d'interprétation X3D utilise la bibliothèque CyberX3D[1] pour réaliser l'analyse syntaxique des fichiers X3D. Le module d'adaptation de X3D s'appuie sur la représentation du graphe de scène de format générée par CyberX3D pour réaliser son adaptation par le SGA. Le tableau 6.7 donne les différentes fonctionnalités de X3D qui ont été implémentées dans notre module d'adaptation X3D. Le module d'adaptation X3D tient compte des fonctionnalités essentielles offertes par X3D pour décrire les objets et la scène. Nous sommes donc en mesure d'adapter, dans différents graphes de scène de moteur, l'ensemble des caractéristiques de base que X3D permet de décrire pour un objet et une scène 3D. Nous avons également pris en compte les nœuds X3D qui permettent d'améliorer les performances de rendu de la scène : le nœud `LOD` pour les niveaux de

1. `www.cybergarage.org/twiki/bin/view/Main/CyberX3DForCC`

détails, le nœud `Switch` pour l'occultation d'une partie du graphe de scène, les champs `DEF` et `USE` pour la réutilisation de sous-graphes de scène ainsi que le nœud `Inline` pour le référencement de fichiers externes. Le module d'adaptation X3D prend également en compte les fonctionnalités d'interactions offertes par X3D : les événements déclenchés par les nœuds `Sensor`, le chaînage d'événements décrit par les nœuds `ROUTE` mais aussi les animations décrites à l'aide des nœuds `Interpolator`.

			MA X3D	MA Collada
DESCRIPTION	Scène	Navigation	X	-
		Point de vue	✓ Viewpoint	X
		Environnement	✓ Background	-
		Éclairage	X	X
		Audio	X	-
	Objet	Géométries	✓ Sphere, ✓ Box, ✓ Cylinder, ✓ IFS	✓ Mesh
		Shaders	X	X
		Matériaux	✓ DiffuseColor, ✓ SpecularColor, ✓ EmissiveColor	✓ Material
		Textures	✓ Texture	✓ Image
		Performance	✓ LOD, ✓ Switch, ✓ Inline, ✓ DEF/USE	✓ URIs
INTERACTION		Animation	✓ Interpolators	-
		Événements	✓ Sensors	-
		Scripts	X	-
		Comportements	✓ Routes	-
		Physique	-	✓ PhysicsScene
		Articulations	X	X

FIGURE *6.7 – Les modules d'adaptation de format de notre implémentation : les nœuds des formats pris en compte.*

6.2.1.2 Instanciation du module d'adaptation Collada

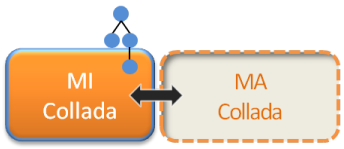


FIGURE *6.8 – L'instance de Collada : son module d'interprétation (MI) et son module d'adaptation (MA).*

Notre second format, le format Collada, est lui aussi libre de droit et se base également sur une syntaxe XML. Il présente également plusieurs caractéristiques intéressantes dans le cadre de l'évaluation du SGA notamment le fait qu'il permette de décrire des propriétés physiques. De plus, c'est un format relativement populaire notamment dans le secteur du jeu vidéo, plusieurs outils et bibliothèques ainsi qu'une large communauté sont donc disponibles pour l'analyse des fichiers Collada.

Le module d'interprétation Collada utilise quant à lui la bibliothèque FCollada [2] pour l'analyse des fichiers Collada mais aussi partiellement la bibliothèque ColladaDOM [3]. Comme le montre le

2. `http://collada.org/mediawiki/index.php/FCollada`
3. `http://collada.org/mediawiki/index.php/COLLADA_DOM`

tableau 6.7, le module d'adaptation Collada, en se reposant sur le graphe de scène de format du module d'interprétation, couvre un grand nombre de fonctionnalités du format Collada. Tout d'abord, il gère les propriétés qui permettent de décrire un objet dans Collada : la géométrie avec le nœud `Mesh` et ses descendants, les matériaux du nœud `Material` et les textures contenues dans le nœud `Image`. Le module d'adaptation gère également les références à des fichiers externes spécifiés avec le mécanisme des URIs (Uniform Ressource Identifier) utilisé par Collada. Enfin, le module d'adaptation tient compte des propriétés physiques du monde et des objets pourvus par Collada.

6.2.1.3 Remarques sur les instances de module d'adaptation de format

La mise en œuvre d'un module d'adaptation de format nécessite une bonne connaissance des spécifications de celui-ci. C'est en effet au niveau du module d'adaptation que l'adaptation proprement dite est faite ; sa conformité aux spécifications garantit la préservation des fonctionnalités et de l'intégrité des modèles. Dans le cas présent, la mise en œuvre des modules d'adaptation diffère en plusieurs points pour les deux formats choisis, aussi bien au niveau de la tâche d'adaptateur que de la tâche d'adapté. Pour les méthodes qui relèvent de la tâche d'adaptateur, les principales différences sont :

- les valeurs par défaut : par exemple la brillance est fixée à 0,2 dans X3D alors que dans Collada, elle est fixée à 10,
- les unités de valeurs : par exemple les rotations sont exprimées à l'aide d'un vecteur et d'un angle dans X3D alors qu'en Collada, les rotations sont exprimées à l'aide d'un quaternion,
- les conventions de repères : par exemple pour les images, le coin de référence est le bord en bas à gauche de l'image pour X3D alors que pour Collada c'est le coin supérieur gauche.

Pour les méthodes qui relèvent de la tâche d'adapté, la différence la plus notable est le nombre de méthodes de l'interface de programmation d'adaptation de formats implémentées. Le module d'adaptation X3D, dans sa tâche d'adapté, implémente toutes les méthodes de l'interface de programmation d'adaptation de format (cf paragraphe 5.2.1.1) alors que le module d'adaptation Collada n'implémente pas les méthodes qui concernent les événements de trames et les événements utilisateurs. La figure 6.9 résume ces différences.

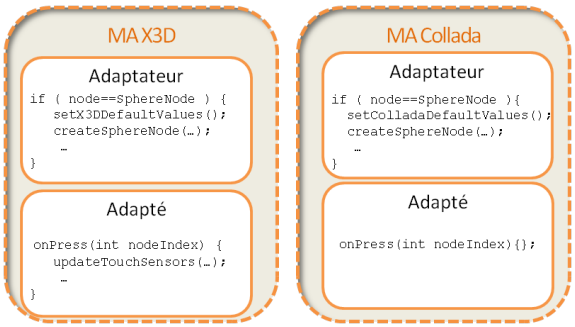


FIGURE *6.9 – Les différences de mise en œuvre des modules d'adaptation de format de notre implémentation.*

6.2.2 LES MOTEURS

Afin de prendre en charge toutes les fonctionnalités proposées par nos formats de test, nous utilisons deux moteurs de types différents pour le rendu de notre application : un moteur de rendu et un moteur physique. Nous avons choisi Ogre[4] comme moteur de rendu et Bullet[5] comme moteur physique.

4. `http://www.ogre3d.org/`
5. `http://bulletphysics.org`

6.2.2.1 Instanciation du module d'adaptation Ogre

Ogre est un moteur de rendu libre de droit qui prend en charge de nombreuses fonctionnalités avancées de rendu. De plus, son interface de programmation est bien documentée et il possède une communauté d'utilisateurs très active, c'est pourquoi nous avons opté pour ce moteur de rendu. Les figures 6.10 et 6.11 montrent les choix d'adaptation des graphes de scène X3D et Collada en graphe de scène Ogre.

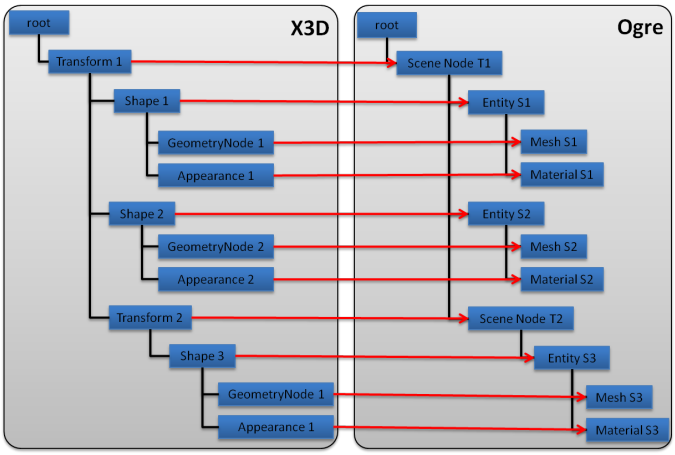


FIGURE *6.10 – Exemple simplifié d'adaptation d'un graphe de scène X3D en un graphe de scène Ogre.*

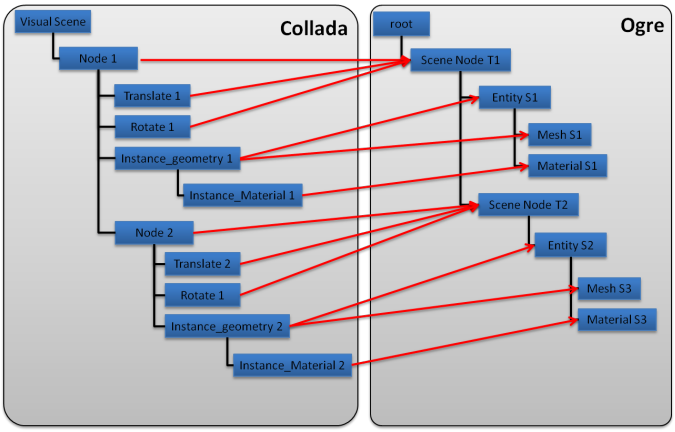


FIGURE *6.11 – Exemple simplifié d'adaptation d'un graphe de scène Collada en un graphe de scène Ogre.*

Ces choix sont mis en place au sein des modules d'adaptation ; un premier choix d'adaptation est fait du côté des modules d'adaptation de format et un second choix est fait du côté du module d'adaptation Ogre. Il est possible d'envisager des versions alternatives des modules d'adaptation qui aboutiront à une autre adaptation des graphes de scène de format en graphe de scène de moteur. Le graphe de scène de Ogre est en effet relativement flexible (cf 6.12), il autorise des sous-entités et des sous-maillages par exemple, possibilité que nous n'avons pas utilisée dans la mise en œuvre de notre module d'adaptation Ogre.

Dans notre implémentation, le moteur Ogre prend en charge l'intégralité du rendu visuel de la scène et capte les événements utilisateurs qu'il transmet ensuite au SGA via l'application. De plus, nous avons choisi de synchroniser les autres graphes de scène de moteur sur la boucle de rendu de Ogre. Par conséquence, les pas de la simulation calculée par Bullet sont synchronisés

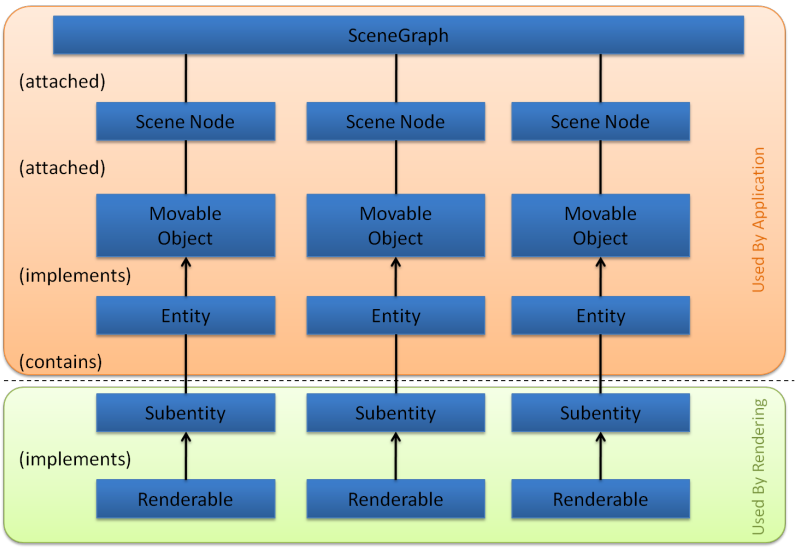


FIGURE *6.12 – La structuration du graphe de scène de Ogre (source : [Jun06]).*

sur la boucle de rendu de Ogre.

6.2.2.2 Instanciation du module d'adaptation Bullet

Le choix de Bullet a été fait pour des raisons similaires à celles évoquées pour Ogre : le fait qu'il soit libre de droit, sa documentation complète et une communauté relativement large. La structuration du graphe de scène de Bullet est, comme la majorité des graphes de scène de moteur physique, limitée à un seul niveau d'enfants. Il n'offre donc pas autant de souplesse que le graphe de scène de Ogre. Cela entraîne dans l'adaptation des graphes de scène X3D et Collada quelques particularités qui seront développées dans le paragraphe 6.3. Le module d'adaptation Bullet, à son initialisation, lance la création d'un monde standard, c'est-à-dire avec une gravité suivant l'axe Y et une détection des collisions paramétrée avec les valeurs par défauts de Bullet.

6.2.3 L'APPLICATION DE TEST

L'application utilisée pour les tests est réduite à un simple programme qui prend en paramètres les adresses des fichiers qu'elle va charger ainsi que les noms des modules d'adaptation de moteur utilisés pour le rendu. Le chargement des modules d'adaptation de moteur se fait en effet dynamiquement au démarrage de l'application. Nous pouvons ainsi aisément choisir d'ajouter ou supprimer un moteur à chaque exécution de l'application de test. Les modules d'adaptation de formats ne sont quant à eux chargés que s'ils sont nécessaires au moment du chargement d'un fichier. Si plusieurs fichiers du même format sont chargés, le module d'adaptation de format correspondant ne sera chargé qu'une seule fois pour le premier fichier de ce format. La figure 6.13 illustre l'architecture de notre instanciation ainsi que l'application qui nous a permis d'effectuer notre évaluation.

6.3 ÉVALUATION DU SGA AU TRAVERS DE NOTRE INSTANCIATION

Nous discutons dans cette partie des différentes interopérabilités offertes par le SGA. Tout d'abord l'interopérabilité entre les formats et l'application, puis l'interopérabilité entre les moteurs et l'application et enfin l'interopérabilité entre les formats et les moteurs.

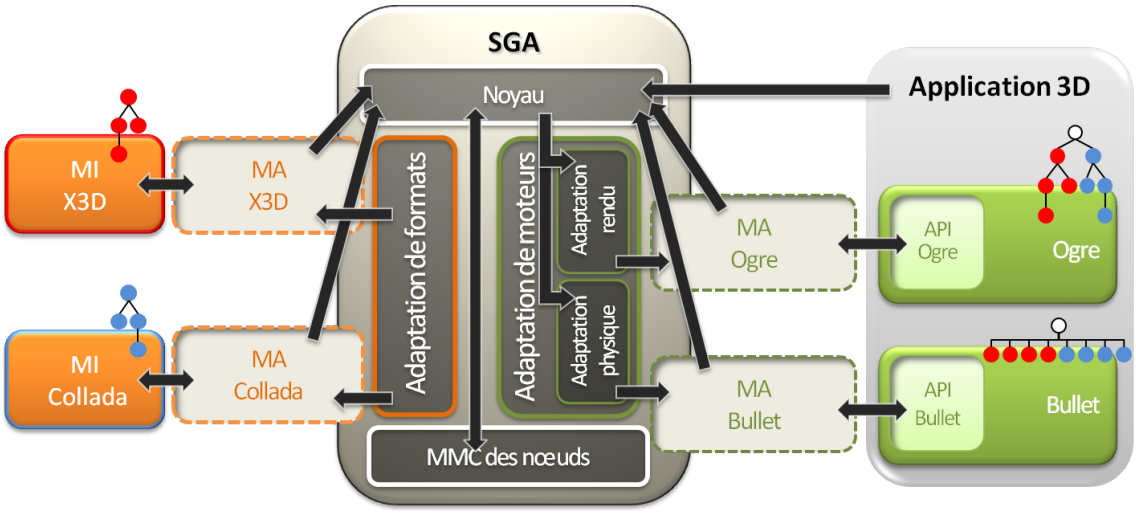


FIGURE 6.13 – *Notre implémentation du SGA et nos instanciations.*

6.3.1 INTEROPÉRABILITÉ DE X3D ET COLLADA AVEC L'APPLICATION

FIGURE 6.14 – *Interopérabilité de X3D et Collada avec l'application*

Notre instanciation du SGA, permet de charger dans notre application de test n'importe quel fichier X3D et n'importe quel fichier Collada et de visualiser l'ensemble de ces fichiers simultanément dans une fenêtre de rendu. Non seulement le chargement des fichiers se fait sans conversion ni modification préalable mais en plus, nous sommes en mesure de nous assurer que l'ensemble des propriétés décrites dans les fichiers sources seront préservées et exploitables par l'application de rendu. Nous pouvons ainsi visualiser les modèles décrits dans leur format d'origine sans erreur mais aussi prendre en compte les fonctionnalités avancées propres au format. Notre application peut ainsi charger des modèles X3D et des modèles Collada avec toutes leurs propriétés. C'est pourquoi l'interactivité des scènes X3D est prise en charge tout autant que les propriétés physiques des modèles Collada.

Notre application pourrait par ailleurs charger d'autres formats, pour cela, il suffit de disposer du module d'interprétation et de son module d'adaptation correspondant sans que cela nécessite de modifier l'application ni de s'assurer de la compatibilité du format avec les composants de rendu sur lesquels se repose l'application. Si ce nouveau format intègre des fonctionnalités non encore gérées par le SGA, il est possible d'étendre l'interface de programmation du SGA pour les prendre en compte. L'interface de programmation du SGA, dans son état actuel, prend en charge un grand nombre d'opérations d'adaptation (création, sélection, modification et suppression de tout types de nœuds). Il est souvent inutile d'ajouter de nouvelles méthodes dans l'interface de programmation pour prendre en charge une nouvelle fonctionnalité, celle-ci peut en effet généralement être décomposée en plusieurs des opérations d'adaptation élémentaires prévues dans l'interface de programmation pour permettre l'adaptation de cette fonctionnalité dans les graphes de scènes de moteur.

Nous avons la possibilité à travers notre implémentation de mixer du X3D et du Collada dans une unique scène mais surtout, de mixer leurs fonctionnalités. L'exemple suivant montre une utilisation de notre application et les capacités de celle-ci.

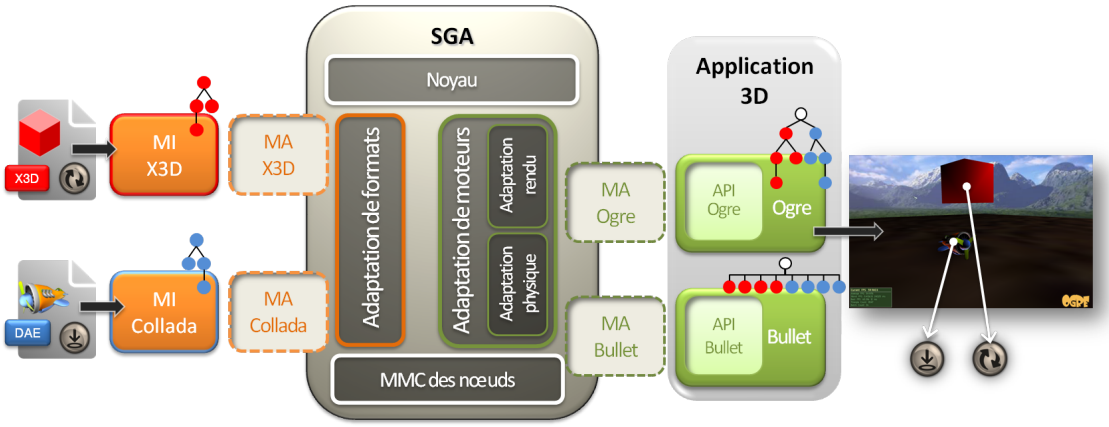


FIGURE *6.15* – *Exemple de scène visualisable dans notre application de test.*

Exemple 6.1 *La figure 6.15 illustre un exemple qui teste l'interopérabilité de l'application avec X3D et Collada.*

En entrée :
- *un fichier X3D qui décrit un cube rouge placé au-dessus d'un sol. Ce cube tourne sur lui-même lorsque l'utilisateur clique dessus,*
- *un fichier Collada qui décrit un avion. Ce dernier possède une masse.*

En sortie : *Le comportement et l'animation du cube rouge sont visibles dans la fenêtre de rendu Ogre. De plus, dans la fenêtre de rendu de notre application, l'avion tombe sur le sol X3D, sa trajectoire étant calculée par Bullet.*

Fonctionnement : *La figure 6.16 montre les messages échangés entre le noyau, les modules d'adaptation et leurs modules d'interprétation lors du chargement du fichier X3D (f_1) et du fichier Collada (f_2).*

Après cette phase de chargement des fichiers, une fois la scène affichée, l'utilisateur peut cliquer dans la scène mais seul un clic sur le cube déclenchera une action. C'est ce que nous montre la figure 6.17 avec un clic sur le cube X3D (n_1) et un clic sur l'avion Collada (n_2) relayé par le noyau du SGA respectivement au module d'adaptation X3D et au module d'adaptation Collada.

6.3.2 INTEROPÉRABILITÉ DES MOTEURS OGRE ET BULLET AVEC L'APPLICATION

Les moteurs de notre application, Ogre et Bullet, au travers du SGA, travaillent conjointement dans leur tâche de rendu de la scène. Ce sont les modules d'adaptation de moteur qui rendent cette communication possible, cela sans avoir besoin d'établir un protocole de communication entre eux. Ceci permet d'utiliser n'importe quel composant de rendu sans se soucier de sa compatibilité avec l'application mais aussi selon les besoins, de changer ou de se passer d'un moteur donné. Bullet et Ogre échangent des informations au cours du déroulement de l'application dans le but de maintenir la cohérence de leurs graphes de scène respectifs. Par exemple, lorsque Bullet au cours de sa simulation modifie la position d'un nœud, il en informe les autres graphes de scène à l'aide du SGA et notamment le graphe de scène de moteur. Le figure 6.19 montre les messages échangés pour mettre en œuvre cette communication entre Ogre et Bullet.

Bullet et Ogre peuvent également collaborer à travers le SGA afin de faciliter la mise en œuvre de certaines opérations. En effet, certaines modifications d'un graphe de scène de moteur nécessitent des informations sur la scène dont ne dispose pas le moteur. Dans ce cas, le SGA peut servir

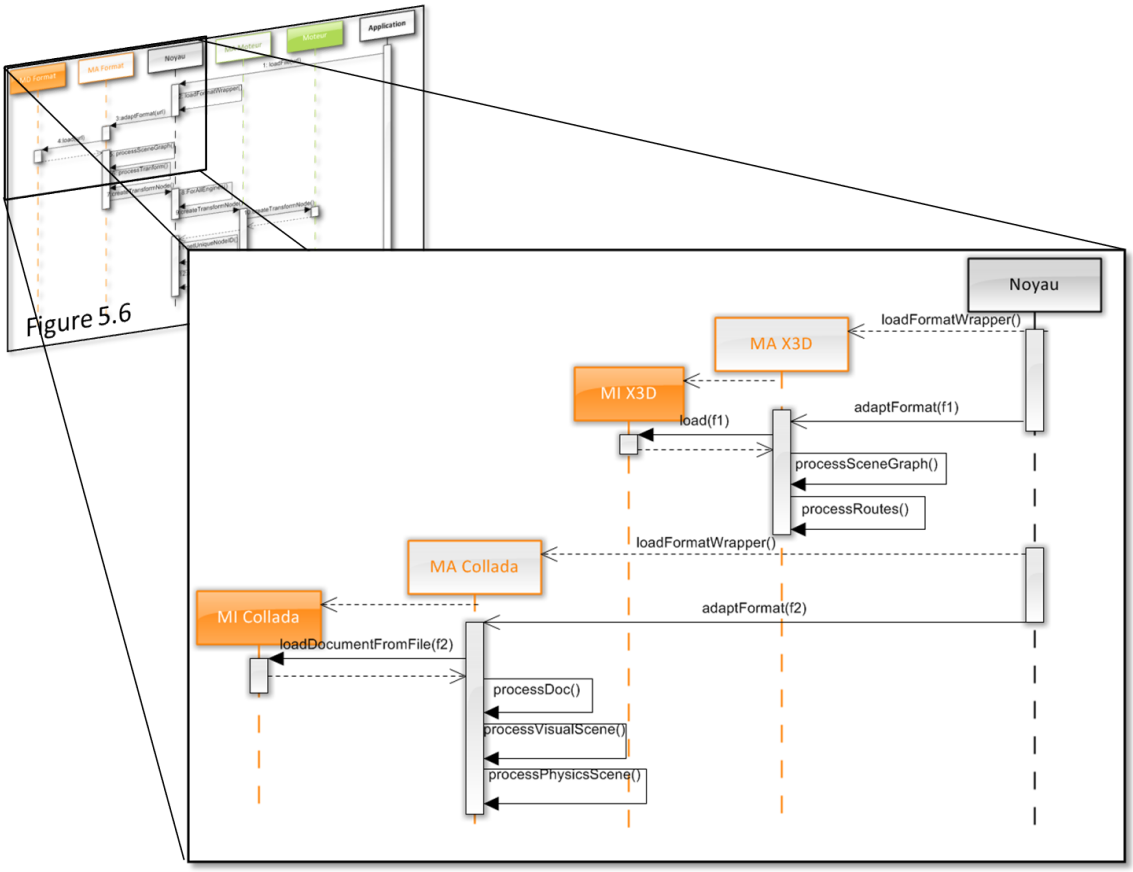


FIGURE *6.16* – *La phase de chargement des fichiers de l'exemple 6.1.*

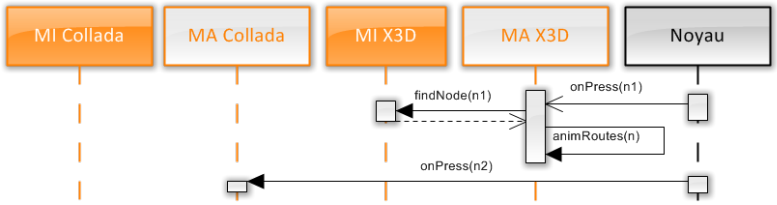


FIGURE *6.17* – *Gestion du clic sur les modèles de la scène de l'exemple 6.1.*

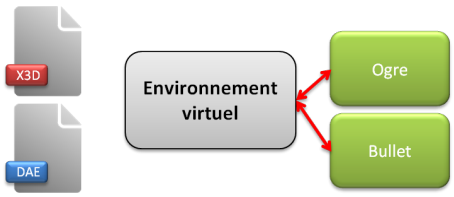


FIGURE *6.18* – *Interopérabilité des moteurs Ogre et Bullet avec l'application*

d'intermédiaire pour relayer les informations utiles d'un moteur à un autre. Par exemple, dans notre implémentation, une collaboration se fait au travers de la boucle de rendu de Ogre. En effet, celle-ci est utilisée pour synchroniser les étapes de la simulation opérée par Bullet comme le

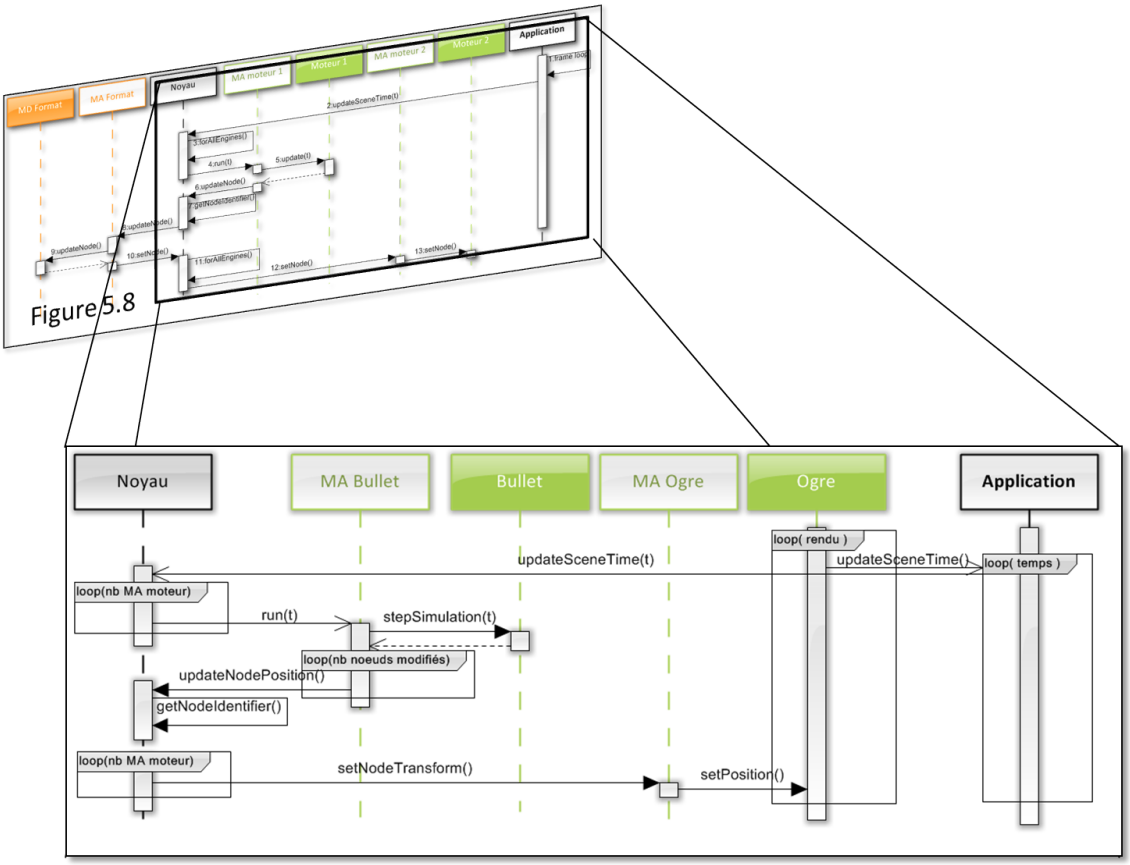


FIGURE *6.19 – Exemple de messages échangés lors d'une modification d'un nœud à l'initiative de Bullet.*

montre la figure 6.19. Le SGA transmet l'information du temps écoulé, gérée par Ogre, à Bullet.

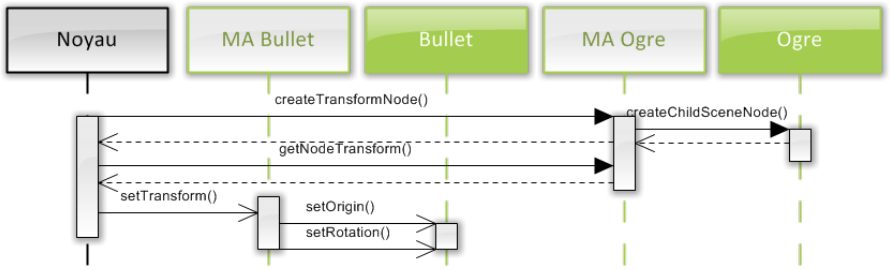


FIGURE *6.20 – Le renseignement des positions globales par Ogre pour Bullet.*

Un autre exemple de collaboration se fait lorsqu'un nouveau nœud de positionnement est créé par le SGA (cf figure6.20). Le noyau du SGA se charge de créer ce nœud à la fois dans le graphe de scène de Ogre et dans le graphe de scène de Bullet. Ogre, comme X3D et Collada, utilise un graphe de scène à plusieurs niveaux de descendance alors que Bullet gère un graphe de scène à un seul niveau. Les positions qu'il manipule sont donc toutes des positions globales dans le monde. Les positions fournies par les modules d'adaptation de format au noyau du SGA étant des positions relatives, le noyau consulte Ogre pour connaître la position globale avant de créer le nœud correspondant dans le graphe de scène de Bullet.

6.3.3 INTEROPÉRABILITÉ ENTRE LES FORMATS ET LES MOTEURS

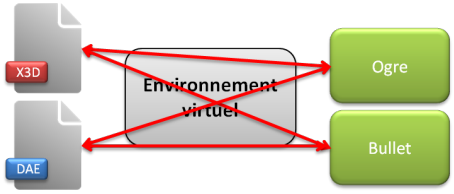


FIGURE 6.21 – *Interopérabilité entre les formats et les moteurs*

Notre application peut donc charger du X3D et du Collada sans problème de compatibilité et utiliser Ogre et Bullet conjointement pour son rendu. Notre implémentation du SGA et l'instanciation que nous en avons fait montrent une interopérabilité entre les contenus et l'application ainsi qu'une interopérabilité entre les moteurs et l'application. On voit, de plus, que cette compatibilité avec X3D et Collada est indépendante de l'utilisation de Ogre et Bullet par l'application. Ogre et Bullet ne permettent pas nativement d'importer du X3D ou du Collada ; c'est l'utilisation du SGA qui le permet dans notre application. Il existe des plugins Ogre et Bullet pour importer du Collada mais le SGA rend possible cette importation sans ces plugins. Nous avons donc une interopérabilité entre les contenus et les moteurs.

6.3.4 EXEMPLES D'UTILISATION

Dans la suite de cette partie, nous allons présenter trois exemples d'utilisation de notre application qui illustrent les possibilités offertes par cette interopérabilité. Nous détaillerons pour chacun les mécanismes mis en œuvre par le SGA pour les prendre en charge. Le premier exemple montre la façon dont sont gérées au travers du SGA les interactions et les animations que l'on peut décrire dans le format X3D. Le second exemple détaillera la façon dont sont prises en charge les propriétés physiques que permettent de décrire le format Collada. Enfin, le dernier exemple montrera les possibilités qu'offre le SGA pour combiner les fonctionnalités propres à chaque format afin de réaliser des scènes 3D complexes.

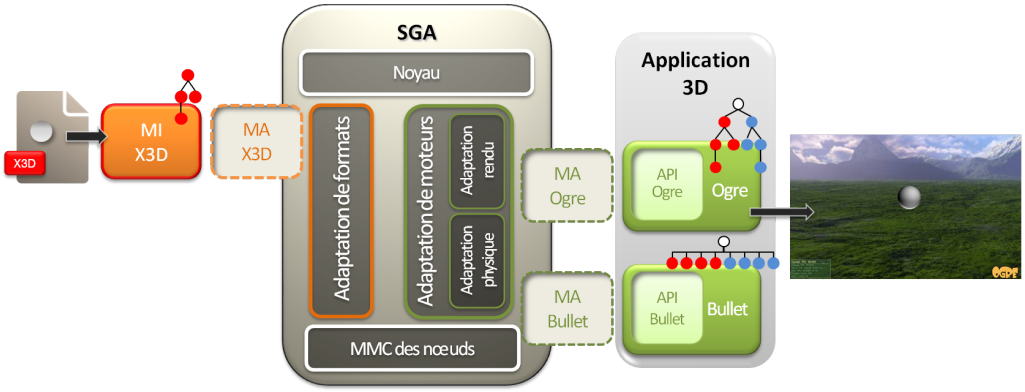


FIGURE 6.22 – *Illustration de l'exemple 6.2*

Exemple 6.2 *L'exemple décrit une sphère dont la transparence change au cours d'un cycle à 5 pas de temps (cf figure 6.22).*

En entrée : *Un fichier X3D qui décrit une sphère (cf table 6.1). Il utilise un nœud* `TimeSensor` *pour gérer le temps et un nœud "*`ROUTE`*" qui lie la valeur du pas du cycle de temps et la valeur de la transparence de la sphère.*

En sortie : *Nous pouvons visualiser la sphère dont la transparence varie en fonction du temps.*

Fonctionnement : *La figure 6.23 indique les messages échangés au cours du déroulement de l'application pour visualiser ce comportement. Dans notre implémentation, le temps de l'application se synchronise sur la boucle de rendu de Ogre, ainsi, à chaque pas de la boucle de rendu, Ogre en informe l'application (message 1). L'application transmet ensuite la mise à jour du temps au noyau du SGA (message 2) qui propage cette information à tous les modules d'adaptation de format (boucle 3) dont le module d'adaptation X3D (message 4). Le module d'adaptation X3D en informe à son tour son décodeur qui va prendre en charge son interprétation selon les propriétés du format X3D. Dans notre exemple, cela se traduit par une mise à jour de la valeur du champ "`value-changed`" du nœud `TimeSensor`. Comme celle-ci est liée par un nœud `ROUTE` à la valeur de la transparence du matériau de la sphère, le nœud matériau de la sphère va être modifié dans le graphe de scène de X3D. Une fois que le décodeur a terminé sa tâche d'interprétation, le module d'adaptation reprend la main. Pour tous les nœuds qui ont été modifiés (boucle 6) dans le graphe de scène de format, il propage cette modification aux graphes de scène de moteur via le noyau. Il va donc informer le noyau que le nœud matériau de la sphère a été modifié (message 7). Le noyau va à son tour propager cette modification à tous les graphes de scène de moteur (boucle 8) et notamment à ceux concernés par cette modification, dont Ogre (message 9). Le module d'adaptation de Ogre se charge alors de faire les modifications correspondantes dans le graphe de scène de Ogre (messages 10).*

```
<Scene>
 <Transform translation='3 3 3' scale='5 5 5'>
   <Shape>
         <Sphere/>
          <Appearance>
                <Material DEF='MAT'/>
          </Appearance>
        </Shape>
 </Transform>
 <TimeSensor DEF='TIMER' loop='true' cycleInterval='5' />
 <ROUTE fromNode='TIMER' fromField='fraction_changed' toNode='MAT' toField='
    transparency' />
</Scene>
```

TABLE *6.1 – Le fichier X3D de l'exemple 6.2.*

Exemple 6.3 *Le fichier Collada de cet exemple décrit deux dés dont l'un a des propriétés physiques (cf figure 6.24).*

En entrée : *Un fichier Collada qui décrit deux dés, l'un d'eux possède une masse et un coefficient de restitution.*

En sortie : *Le SGA permet d'exploiter ces propriétés offertes par le format Collada et ainsi d'en tenir compte dans le rendu du modèle. Lorsque nous chargeons ce fichier dans notre application, nous pouvons voir le dé avec une masse tombe alors que l'autre reste statique.*

Fonctionnement : *La figure 6.25 indique les messages échangés afin de prendre en compte ces propriétés dans le rendu de la scène. Après le chargement du fichier Collada par l'application, celle-ci délègue sa gestion au module d'interprétation de Collada puis le module d'adaptation de format prend le relais pour procéder à l'adaptation du graphe de scène de format. Le module d'adaptation Collada va donc commencer l'adaptation de la scène physique du fichier et entamer le traitement des instances de modèle physique (message 1). Pour chaque instance, il transmet au noyau les propriétés physiques à attribuer au nœud (message 2). Le noyau aidé de l'index de nœuds retrouve tous les nœuds correspondants dans les graphes de scène de moteur. Il consulte Ogre pour connaître la position globale du nœud (message 3) et une fois cette information obtenue, il transmet au module d'adaptation de Bullet les propriétés physiques du nœud (message 4). Le module d'adaptation de Bullet transcrit ensuite cette requête en instructions pour Bullet (messages 5*

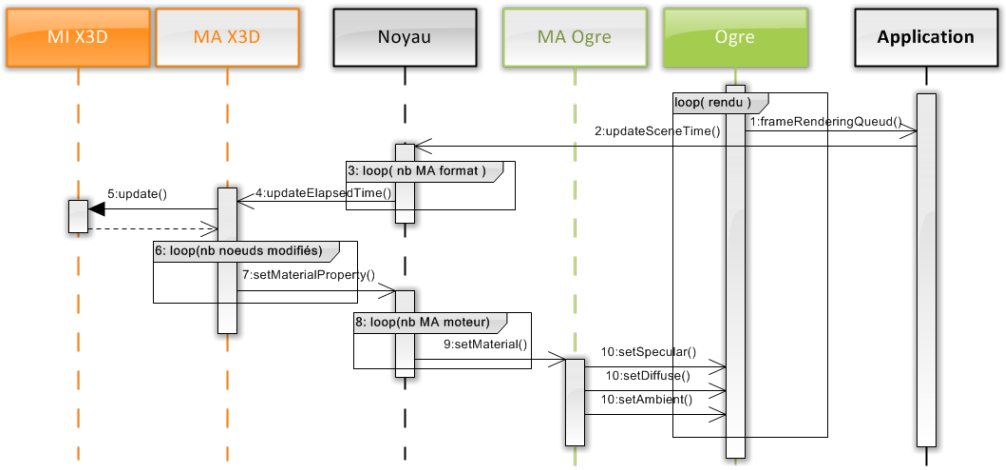


FIGURE *6.23* – *Les messages échangés par les composants du SGA pour gérer le fichier X3D de l'exemple 6.2*

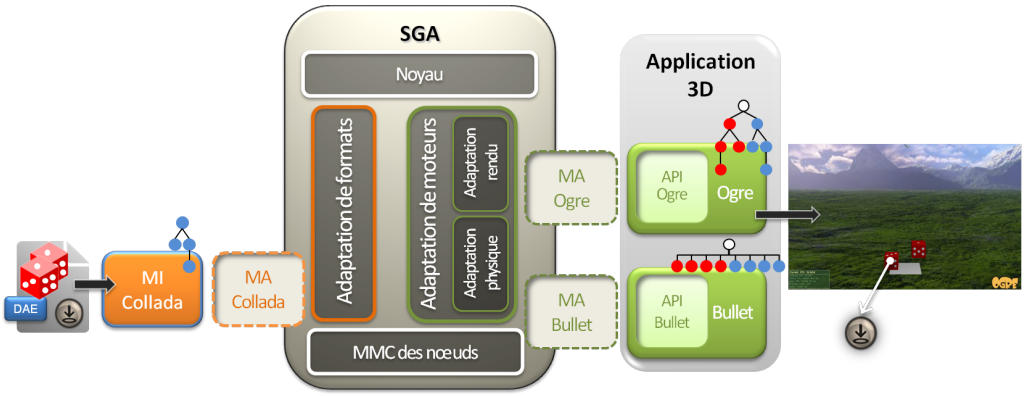


FIGURE *6.24* – *Illustration de l'exemple 6.3*

à 8). À l'issue de l'adaptation de la scène physique du fichier, Bullet est en mesure d'effectuer sa simulation et de calculer la position du dé doté d'une masse à chaque trames de rendu. Pour chaque nouvelle position calculée, il transmet via le SGA la nouvelle position du dé à son graphe de scène de format puis le module d'adaptation de format propage cette mise à jour à Ogre pour qu'à son tour il modifie la position du dé dans son graphe de scène de moteur. Ce processus est illustré par la figure 5.8.

Exemple 6.4 *Ce dernier exemple montre les possibilités offertes par le SGA grâce à ses capacités d'interopérabilité (cf figure 6.26).*

En entrée : *Un fichier X3D (cf table 6.3) constitué d'un nœud de positionnement qui a pour enfant un nœud* `TouchSensor` *permettant de capter les évènements sur tous les enfants de son nœud parent. Il a pour frère un nœud* `Inline` *qui référence un fichier externe et qui est en l'occurrence un fichier Collada qui décrit un canard. Dans la suite du fichier X3D, on trouve plusieurs nœuds* `ROUTE` *qui ont pour rôle d'animer les sous-graphes de scène enfants du nœud* `Transform` *de la ligne 3.*

En sortie : *Nous pouvons voir dans la fenêtre de rendu de Ogre, le canard tourner sur lui-même.*

Fonctionnement : *Lorsque le module d'adaptation X3D atteint le nœud* `Inline`*, il appelle le noyau du SGA pour lui demander le chargement du fichier Collada. Celui-ci charge alors le module d'adaptation approprié qui gère alors l'adaptation du graphe de scène du fichier comme présenté dans la figure 5.14. Une fois les deux graphes de scène affichés dans la fenêtre de rendu de Ogre, lorsque l'utilisateur clique sur*

```
...
<library_physics_materials>
  <physics_material id="pCube1-PhysicsMaterial" name="pCube1-PhysicsMaterial">
    <technique_common>
          <dynamic_friction>0</dynamic_friction>
      <restitution>0.2</restitution>
      <static_friction>0</static_friction>
        </technique_common>
  </physics_material>
        ...
</library_physics_materials>
<library_physics_models>
  <physics_model id="pCube1-PhysicsModel" name="pCube1-PhysicsModel">
    <rigid_body name="pCube1-RigidBody" sid="pCube1-RigidBody">
          <technique_common>
                <instance_physics_material url="#pCube1-PhysicsMaterial"/>
        <shape>
          <box>
            <half_extents>10 10 10</half_extents>
          </box>
        </shape>
        <dynamic>false</dynamic>
        <mass>1.0</mass>
      </technique_common>
        </rigid_body>
  </physics_model>
  ...
</library_physics_models>
<library_physics_scenes>
  <physics_scene id="Scene-Physics" name="Scene-Physics">
    <instance_physics_model url="#pCube1-PhysicsModel">
          <instance_rigid_body body="pCube1-RigidBody" target="#pCube1"/>
        </instance_physics_model>
    <instance_physics_model url="#pCube2-PhysicsModel">
      <instance_rigid_body body="pCube2-RigidBody" target="#pCube2"/>
    </instance_physics_model>
    <instance_physics_model url="#Plane-PhysicsModel">
      <instance_rigid_body body="Plane-RigidBody" target="#Plane"/>
    </instance_physics_model>
        <technique_common>
          <gravity>0 -9.810000 0</gravity>
               <time_step>0.0166666</time_step>
        </technique_common>
  </physics_scene>
</library_physics_scenes>
...
```

TABLE *6.2 – Un extrait du fichier Collada de l'exemple 6.3.*

le canard, celui-ci tourne selon l'animation décrite dans le fichier X3D. La figure 6.27 montre les messages échangés par les composants du SGA pour réaliser cette opération. Pour cela, Ogre transmet à l'application l'évènement du clic avec la position du pointeur de la souris (message 1). L'application transmet ensuite au noyau l'évènement avec l'identifiant du nœud cliqué dans le graphe de scène de Ogre (message 2). Le noyau consulte ensuite l'index de nœuds (message 3) pour retrouver le module d'adaptation de format du nœud cliqué. Il transmet ensuite l'évènement et les informations relatives au module d'adaptation de format

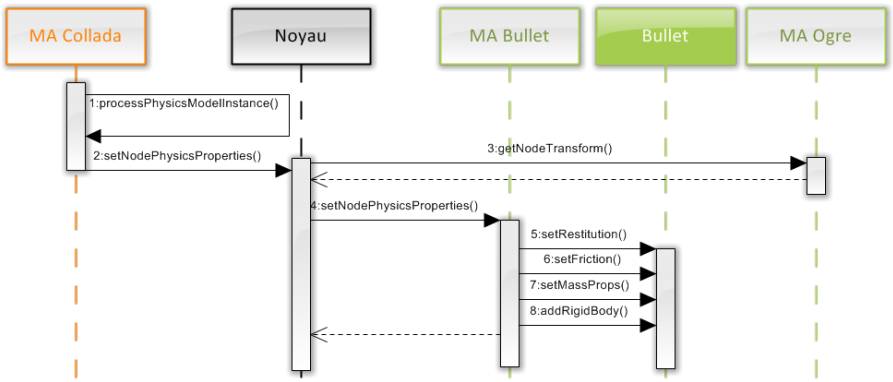


FIGURE 6.25 – *Les messages échangés par les composants du SGA pour prendre en compte les propriétés physiques décrites dans le fichier Collada de l'exemple 6.3.*

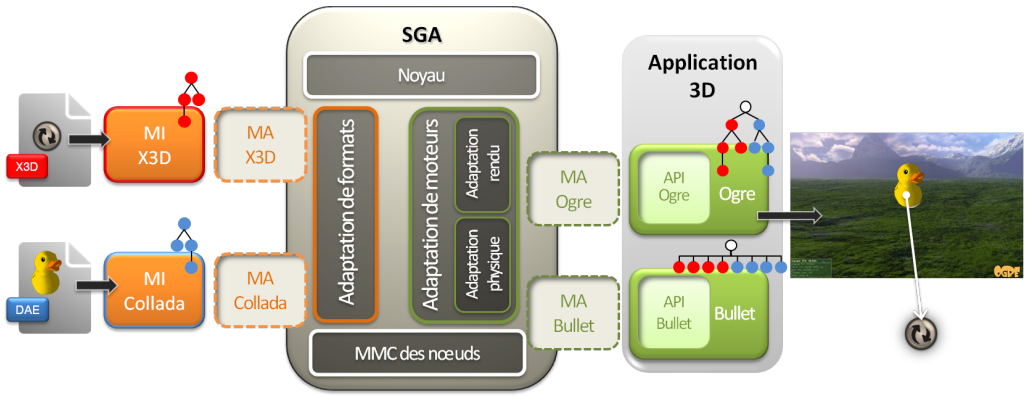


FIGURE 6.26 – *Illustration de l'exemple 6.4*

(message 4) qui délègue au module d'interprétation de format l'interprétation de cet évènement (message 5). Le résultat de cette interaction est ensuite transmit au module d'adaptation de format puis au noyau (boucle 6 et message 7) qui informe ensuite les graphes de scène de moteurs des modifications à effectuer (boucle 8 et message 9). Dans notre exemple, l'évènement entraîne une rotation du nœud parent du canard ce qui permet de le faire tourner sur lui-même. Ogre reçoit donc via son module d'adaptation de moteur les instructions pour mettre à jour la position de ce nœud (messages 10 et 11).

6.4 TESTS ET DISCUSSION

6.4.1 DISCUSSION

Notre implémentation nous a permis de faire quelques observations quant aux performances du SGA notamment pour des fichiers de grandes tailles. Nous avons identifié trois problèmes :

1. Le premier problème vient de la recherche dans le module de mise en correspondance des nœuds. La complexité de cette recherche est de l'ordre de $O(\log n)$ où n est le nombre de nœuds dans les graphes de scène de format qui ont une représentation dans au moins un

```
<Scene>

  <Transform DEF='TRANS'  translation='-15 -5 1.0' rotation='0.0 0.707 0.707
      0.9'>
    < Inline url='..\collada\Duck\duck.dae' />
    < TouchSensor DEF='Clicker'/>
  </Transform>

  <TimeSensor DEF='Timer' cycleInterval='4.0' />
  <OrientationInterpolator DEF='Animation' key='0.0 0.5 1.0' keyValue='0.0 0.0
      1.0 0.0 0.0 0.0 1.0 2.1 0.0 0.0 1.0 0.0'/>

  <ROUTE fromField='touchTime' fromNode='Clicker' toField='startTime' toNode='
      Timer' />
  <ROUTE fromField='fraction_changed' fromNode='Timer' toField='set_fraction'
      toNode='Animation'/>
  <ROUTE fromField='value_changed' fromNode='Animation' toField='rotation'
      toNode='TRANS'/>

</Scene>
```

TABLE *6.3 – Extrait du fichier X3D de l'exemple 6.4 : il référence un fichier Collada.*

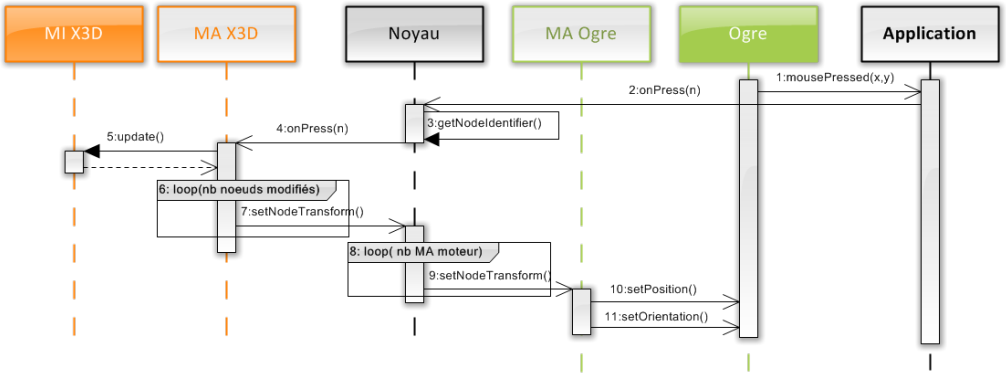


FIGURE *6.27 – Le traitement d'un évènement de clic par le SGA dans le cadre de l'exemple 6.4*

graphe de scène de moteur. Son impact sur les performances du SGA est donc a priori limité. En effet, chaque modification d'un nœud d'un graphe de scène de format nécessite la recherche de jusqu'à n entrées dans l'index de nœuds, n étant le nombre de moteurs utilisés par l'application. Pour un nœud d'un graphe de scène de moteur, on fera la recherche d'une entrée de l'index de nœuds dans un premier temps. Si la modification entraîne une modification du graphe de scène de format alors il faudra faire de nouveau la recherche de n entrées dans l'index de nœuds.

2. Le second problème identifié vient de l'appel additionnel aux méthodes du noyau du SGA à chaque mise à jour d'un graphe de scène ; ceci comparé à un décodeur ou à un moteur traditionnel. Ce point ne semble pas problématique étant donné les capacités et les performances des machines actuelles même sur des scènes de très grande taille.
3. Le dernier point que nous avons identifié provient de la nécessité de maintenir plusieurs représentations des graphes de scène impliqués au cours de l'exécution de l'application. Pour chaque fichier chargé, nous avons en effet une représentation du graphe de scène gérée par

le module d'interprétation de format et une copie (plus ou moins partielle) de celui-ci dans chaque moteur impliqué dans le rendu de la scène. Étant donné les capacités et les performances des machines actuelles, ceci ne devrait avoir que peu d'impact sur le rendu. On peut d'ailleurs noter que ceci se rencontre de plus en plus dans les moteurs 3D car les capacités de stockage et les temps d'accès des machines actuelles le permettent sans que cela soit trop pénalisant pour les performances des moteurs. Les bénéfices obtenus par ailleurs par l'efficacité de la consultation de ces graphes de scène spécialisés compensent largement ce fait. En effet, chaque graphe de scène répond à une fonctionnalité différente et son organisation est pensée de façon à optimiser sa consultation. Cependant, si les scènes chargées sont extrêmement complexes, l'utilisation d'un module de gestion de cache reste fortement conseillée et facilement intégrable à l'architecture SGA présentée. Finalement, l'espace mémoire nécessaire à la mise en oeuvre de l'architecture SGA concerne essentiellement le module de mise en correspondance des nœudsqui est implémenté sous la forme de deux tables (Figure 5.2). Cet espace mémoire peut être considéré comme négligeable vis à vis de l'espace mémoire utilisé par le stockage des contenus des graphes de scène.

Concernant notre implémentation, plusieurs points permettraient de gagner en efficacité. Tout d'abord, nous nous sommes aperçu que le choix des outils du module d'interprétation était crucial et que leurs performances avaient une forte incidence sur le SGA. D'autre part, nous n'avons pas utilisé de géométries optimisées au niveau des moteurs alors que Bullet et Collada le permettent, cela rendrait la simulation de Bullet plus efficace et accélérerait les traitements du SGA. Dernier point, la mise en réseau de notre architecture et l'utilisation de machines distantes pour le découpage des tâches seraient sans doute des optimisations intéressantes.

6.4.2 TESTS

Nous présentons ici le test de six scènes que nous avons chargées dans notre application pour évaluer l'impact du SGA sur le traitement de ces scènes en terme d'utilisation de l'UC. Nous souhaitons à travers ces tests chiffrer l'incidence des points 1 et 2 du paragraphe précédent (cf 6.4.1). Nous n'avons pas jugé nécessaire de tester le point 3 car, comme expliqué dans le paragraphe précédent, le fait d'utiliser plusieurs graphes de scène est une technique relativement courante dans les moteurs actuels.

6.4.2.1 Protocole de test

Les six scènes de test sont présentées dans le tableau 6.4.2.1. Nous avons trois scènes X3D et trois scènes Collada qui ont été choisies dans le but de mesurer l'impact du SGA sur leur rendu.

6.4.2.2 Résultats et interprétation

Nous présentons les résultats de nos tests sous la forme d'un graphique qui montre l'utilisation de l'UC en fonction du temps. Nous avons voulu évaluer à travers ce test, la part d'utilisation de l'UC par le SGA dans le processus de l'application. Nous nous sommes servis de l'outil VTune [6] pour générer ces graphiques, il s'agit d'un outil de profilage.

Test 1

Cette scène ne décrivant pas d'animation ni d'interaction, il n'y a que son chargement qui sollicite le SGA à l'initiative du module d'adaptation du format X3D. Au moment du chargement de la scène, chaque nœud traité par le SGA est référencé dans le module de mise en correspondance des nœuds. Nous pouvons voir ce traitement qui correspond à la zone rouge du graphique. Nous pouvons voir plus loin sur la ligne de temps deux sollicitations du SGA qui correspondent au

6. http://software.intel.com/en-us/intel-vtune-amplifier-xe

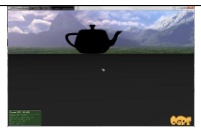
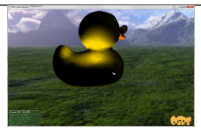
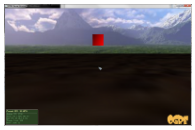
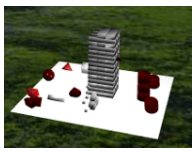
	Format	Taille	Description	Illustration
1	X3D	25.2 Ko	Une scène X3D qui décrit une théière non texturée.	
2	Collada	277 Ko	Une scène Collada qui décrit un canard texturé	
3	X3D	1,95 Ko	Une scène qui décrit un cube, une interaction et une animation : lorsque l'on clique sur le cube, celui-ci fait un tour sur lui-même.	
4	Collada	0.98 Mo	Une scène qui décrit un avion avec des propriétés physiques ; une masse lui est associée.	
5	X3D	20.6 Ko	Une scène qui décrit 50 cubes avec une interaction et une animation : lorsque l'on clique sur l'un des cubes, ils se mettent tous à tourner.	
6	Collada	594 Ko	Un jeu de Jenga™avec des propriétés physiques : chaque pièce du jeu possède une masse et le monde a une gravité.	

TABLE *6.4 – Présentation des six scènes qui nous ont servies pour les tests.*

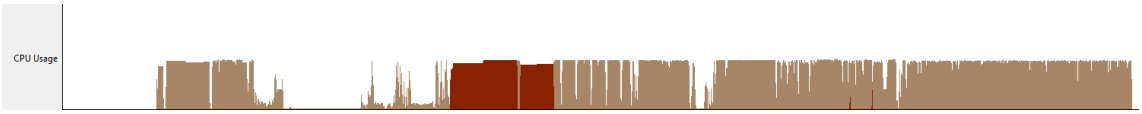


FIGURE *6.28 – Utilisation de l'UC par les processus de l'application et la part du SGA en rouge pour le test 1.*

passage de la souris sur la théière. Dans ce cas, Ogre transmet au SGA l'identifiant du nœud, le SGA interroge ensuite le module de mise en correspondance des nœuds pour retrouver l'identifiant du nœud dans son graphe de scène de format. Comme aucune interaction n'est associée à ce nœud, la sollicitation du SGA s'arrête là.

Test 2

L'interprétation donnée pour le test 1 s'applique également pour ce test. Nous voyons que le SGA réagi de la même façon pour des modèles X3D et des modèles Collada. La seule différence

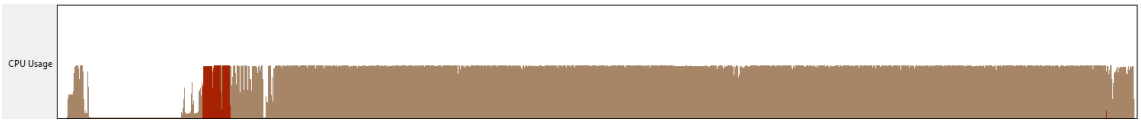


FIGURE *6.29 – Utilisation de l'UC par les processus de l'application et la part du SGA en rouge pour le test 2.*

notable est que le chargement du modèle Collada est plus rapide que celui du modèle X3D car l'analyseur syntaxique utilisé dans le module d'interprétation du format Collada est plus rapide.

Test 3

FIGURE *6.30 – Utilisation de l'UC par les processus de l'application et la part du SGA en rouge pour le test 3.*

Pour ce test, nous pouvons voir encore une fois nettement le chargement du modèle. Nous voyons ensuite plusieurs sollicitation du SGA lorsque le cube est cliqué puis lorsque celui-ci effectue sa rotation. A chaque pas, la nouvelle position du cube est transmise aux moteurs Ogre et Bullet par l'intermédiaire du SGA.

Test 4

FIGURE *6.31 – Utilisation de l'UC par les processus de l'application et la part du SGA en rouge pour le test 4.*

Le test 4 montre encore une fois, le chargement du modèle puis l'utilisation du SGA lorsque l'avion commence sa chute. Pour nos tests, nous avons ralenti la simulation effectuée par Bullet, c'est pourquoi le modèle reste un moment en suspens avant de commencer sa chute. Ceci se traduit sur le graphique par l'écart entre la phase de chargement et la phase de modification. A chaque pas de la simulation de Bullet, le SGA est sollicité deux fois. Une première fois pour transmettre l'information au graphe de scène de format et une seconde fois pour transmettre la nouvelle position aux graphes de scène de moteur.

Test 5

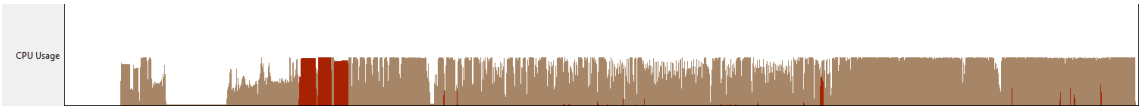


FIGURE *6.32 – Utilisation de l'UC par les processus de l'application et la part du SGA en rouge pour le test 5.*

Ce test montre lui aussi une phase de chargement puis une phase de traitement de l'animation. L'animation de ce test sollicite 50 fois plus le SGA que le test 3 car les 50 cubes de la scène bougent simultanément.

Test 6

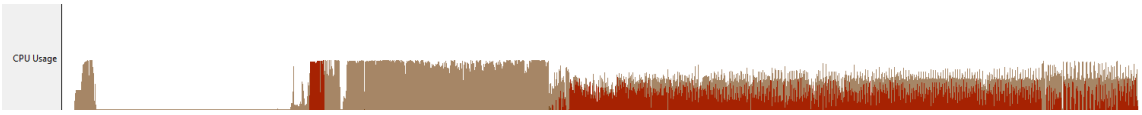


FIGURE *6.33 – Utilisation de l'UC par les processus de l'application et la part du SGA en rouge pour le test 6.*

Ce dernier test montre lui aussi la phase de chargement puis la phase de modification. Entre les deux, nous pouvons voir une phase de latence pour le SGA car le jeu de Jenga™reste un moment en équilibre avant de s'écrouler. Ce test sollicite plus le SGA que le précédent comme le montre le graphique. En effet, a chaque pas de temps, le SGA est sollicité par le module d'adaptation de Bullet pour mettre à jour le graphe de scène de format puis de nouveau par le module d'adaptation du format Collada pour mettre à jour les graphes de scène de moteur.

6.4.2.3 Synthèse

	Pourcentage d'utilisation de l'UC par le SGA
1	12,8%
2	2,8%
3	6,4%
4	1,1%
5	5,8%
6	14,6%

TABLE *6.5 – Synthèse des tests.*

Le tableau 6.4.2.3 donne le pourcentage moyen d'utilisation de l'UC par le SGA. On voit que même pour des scènes qui requièrent beaucoup d'échanges entre les graphes de scène de format et les graphes de scène de moteur, sont impact reste raisonnable.

6.5 SYNTHÈSE ET CONCLUSION

Nous avons montré dans ce chapitre la faisabilité d'une mise en oeuvre de l'architecture SGA à travers l'implémentation que nous en avons faite. Nous avons ensuite réalisé quatre instanciations qui nous ont permis de charger des fichiers X3D et des fichiers Collada dans une application se basant sur les moteurs Ogre et Bullet. Grâce au SGA, nous avons pu visualiser des scènes composées de fichiers X3D et de fichiers Collada sans les modifier tout en conservant leurs propriétés respectives. Ainsi, Ogre à pu gérer et afficher les interactions tout autant que les animations décrites dans des fichiers X3D. Par ailleurs, Bullet a pu prendre en compte les propriétés physique décrites dans des fichiers Collada et collaborer avec Ogre par l'intermédiaire du SGA dans le but de visualiser la simulation calculée par Bullet. Nous avons également démontré que les traitements supplémentaires opérés par le SGA pour le rendu des scènes avaient un impact raisonnable sur les performances de l'application. +

Chapitre 7

VERS LA COMPOSITION DE SCÈNES 3D : UN MODÈLE DE CONTENEUR POUR LES FORMATS 3D

LE SGA nous offre la possibilité de charger dans un environnement virtuel 3D n'importe quel contenu 3D basé graphe de scène quels que soient les composants de rendu utilisés par son application. Il permet de préserver les fonctionnalités de chaque format et les rend exploitables par les moteurs. Le SGA rend possible une interopérabilité entre les formats 3D et les environnements virtuels 3D. Nous avons vu dans le chapitre précédent les possibilités qu'offre l'interopérabilité créée par le SGA, notamment la possibilité de mixer différents formats dans une même scène afin de tirer partie des fonctionnalités proposées par chacun. Nous souhaitions aller plus loin afin d'avoir une interopérabilité entre les formats eux-mêmes. Nous présentons dans cette partie le résultat de cette réflexion et comment elle nous a mené à la conception d'un modèle de conteneur de fichiers 3D. Nous verrons ensuite en détail ce modèle et la façon dont il s'intègre au SGA. Nous terminerons ce chapitre par un exemple détaillé d'utilisation dans notre implémentation du SGA avant de conclure sur une discussion de notre modèle de conteneur.

7.1 UN CONTENEUR DE FICHIERS 3D

Cette partie explique pourquoi nous proposons un conteneur de fichiers 3D comme moyen de rendre interopérables les formats 3D entre eux. Nous y présentons également les critères de conception qui nous paraissent indispensables à ce type de modèle.

7.1.1 VERS L'INTEROPÉRABILITÉ ENTRE LES FORMATS 3D

Le SGA rend interopérables les contenus 3D avec une application en charge d'un environnement virtuel 3D mais il ne permet pas l'interopérabilité entre ces contenus. Nous sommes en effet capable grâce au SGA de créer une scène composée de plusieurs objets encodés dans des formats différents, ils sont alors soumis aux règles d'un même monde virtuel mais nous n'avons pas de moyen de décrire les interactions entre ces objets alors qu'il serait souhaitable de pouvoir le faire.

Pour répondre à ce besoin, nous avons opté pour l'utilisation d'un format conteneur. Un tel format nous permet de lister les fichiers 3D mis en relation et d'y définir les connexions qui les lient. C'est une solution flexible et extensible qui a de plus l'avantage de se conformer au cadre d'étude que nous nous sommes fixé pour les contenus. Elle est en effet compatible avec le plus grand nombre de formats possible et comme elle ne nécessite pas de modifier les fichiers listés, elle préserve l'information contenue dans ces derniers et évite leurs conversions.

7.1.2 LES FORMATS CONTENEURS

Les formats conteneurs sont un modèle utilisé de longue date dans de nombreux domaines de l'informatique. Néanmoins, l'état de l'art nous a appris qu'il n'existait pas de format de ce type pour les fichiers 3D mis à part le format 3DMLW [1]. Ce format prévoit en effet l'inclusion de modèles encodés dans un autre format à l'intérieur de fichiers 3DMLW. Il permet d'inclure des fichiers aux formats 3DS Max, Collada, Object 3D et Blender mais il ne permet pas de décrire des interactions entre eux.

À l'opposé, d'autres domaines sont très prolifiques en la matière comme le domaine des médias numériques. Il existe en effet plus de 60 formats conteneurs dans ce domaine. Ils sont de deux types :

- les formats conteneurs de stockage,
- les formats conteneurs de transport.

L'objectif de ces formats est de proposer un nombre réduit de formats dans un domaine où il en existe une très grande variété. Cette variété s'étend des formats natifs utilisés par les outils de production, aux formats de compression. Dans le domaine des médias numériques, les formats conteneurs permettent de simplifier les tâches courantes telles que l'acquisition, l'enregistrement, la recherche, la consultation, l'édition, la conversion, le transfert ou la distribution des données multimédias. Une des principales fonctionnalités offertes par ces formats est la possibilité d'ajouter des métadonnées qui permettent de contrôler la restitution de leurs contenus. Par exemple, les formats AVI et DIVX permettent d'encapsuler plusieurs pistes audio et vidéo puis de les synchroniser grâce aux métadonnées lors de leurs restitutions. Nous ne pouvons cependant pas calquer directement le modèle des formats conteneurs multimédias à un modèle de conteneur de fichiers 3D car ils ne couvrent pas les mêmes usages. Les formats conteneurs multimédias et 3DMLW ne permettant pas de répondre à notre besoin, nous avons choisi de créer un modèle de format conteneur pour les fichiers 3D.

7.1.3 LES CRITÈRES DE CONCEPTION

Un format conteneur de fichiers 3D n'a pas pour objectif de synchroniser les modèles 3D comme un format conteneur de fichiers multimédias. Il doit plutôt permettre d'organiser les modèles dans la scène 3D finale et de définir des connections entre eux. Nous avons donc cherché à définir les critères de conception d'un conteneur de fichier 3D et avons établi quatre critères :

1. *encapsulation de la majorité des formats* : un format conteneur pour les fichiers 3D doit être capable d'intégrer le plus de formats possible. Comme exposé dans le chapitre 2, la très grande diversité des formats 3D provient du fait que chacun, par les fonctionnalités qu'il propose, répond à un besoin particulier. Il est donc essentiel qu'un format conteneur de fichiers 3D soit capable de prendre en compte cette diversité.
2. *organisation dans l'espace* : un format conteneur pour la 3D va permettre de composer une scène à partir de modèles issus de différents fichiers aux formats hétéroclites. Comme chaque fichier possède son propre repère géométrique, il est indispensable qu'un format conteneur de fichiers 3D permette de positionner dans l'espace les fichiers les uns par rapport aux autres.
3. *intégration de métadonnées* : cette possibilité des formats conteneurs existants permet différents usages intéressants telle que l'annotation des contenus encapsulés pour la restitution ou le traitement automatisé de ces données. Ce critère nous paraît donc important à prendre en compte.
4. *définition d'interaction* : c'est en effet le but de la conception de ce format conteneur, il nous faut mettre en place des conventions pour définir les interactions entre les fichiers mis en relation.

1. www.3dmlw.com

La conception de notre modèle de conteneurs de fichiers 3D, appelé 3DFC, tient compte de ces quatre critères.

7.2 LA CONCEPTION DU FORMAT 3DFC

Dans cette partie, nous allons commenter la conception du 3DFC. Nous verrons comment nous avons tenu compte des critères établis précédemment ainsi quels sont les autres principes qui nous ont guidé.

7.2.1 LE MODÈLE DU FORMAT 3DFC

Nous avons appelé notre modèle de conteneur 3DFC qui signifie "conteneur de fichiers 3D" et qui est l'acronyme de l'équivalent en anglais : 3D Files Container. Notre intention n'est pas de proposer un nouveau format mais plutôt de concevoir un modèle qui pourrait être aisément intégré dans un format existant. Dans cette optique, nous avons opté pour une syntaxe XML qui est un standard répandu et libre de droit. Il est par ailleurs supporté par un grand nombre d'outils et extensible ce qui rend son utilisation et sa maintenance aisées.

Un autre point important de la conception du modèle 3DFC est le fait que nous voulions qu'il soit le plus compact possible. Nous seulement parce qu'il s'agit d'un modèle et non pas d'un nouveau format mais aussi pour qu'il soit plus facilement intégrable dans un format existant. Il se compose donc de peu de nœuds. Nous y trouvons uniquement les nœuds essentiels, les autres fonctions caractéristiques d'un format 3D sont fournies par les formats encapsulés. 3DFC propose deux types de nœuds : des nœuds de description et des nœuds d'interaction.

7.2.1.1 Les nœuds de description

Les nœuds de description que propose le modèle de 3DFC permettent d'identifier et d'annoter les fichiers 3D qui sont combinés dans une même scène 3D. 3DFC possède quatre nœuds pour décrire la scène finale :

- un nœud racine : `SGA-3DContainer`,
- un nœud de regroupement : `Group`,
- un nœud de positionnement : `Transform`,
- et un nœud de contenu : `Content`.

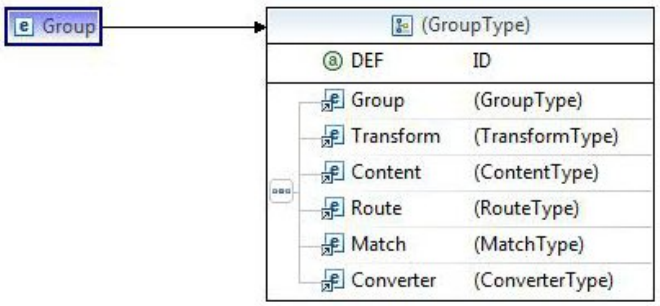


FIGURE *7.1 – Le nœud Group de 3DFC.*

Les nœuds `Group` et `Transform` sont des nœuds courants dans les formats 3D. La figure 7.1 donne les attributs du nœud `Group` et ses enfants admissibles. Le nœud `Transform` permet de définir une position à partir des attributs de translation et de rotation ainsi qu'une homothétie. Ses enfants admissibles sont listés dans la figure 7.2.

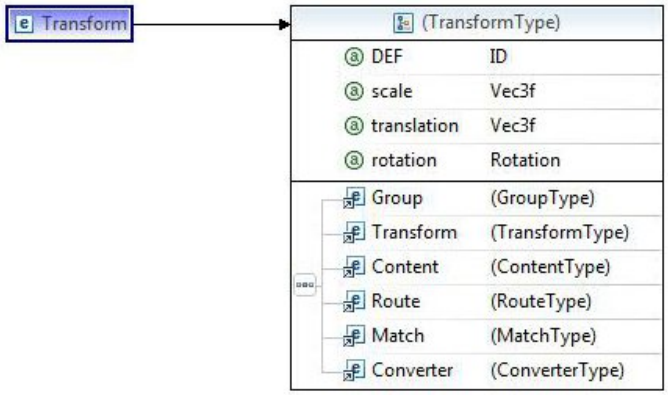


FIGURE *7.2 – Le nœud Transform de 3DFC.*

Le nœud `Content` permet quant à lui de référencer un fichier externe (cf figure 7.3). Ses attributs permettent de préciser son adresse (`url`), son format (`type`), l'adresse d'un module d'adaptation approprié (`wrapper_url`) et l'adresse du module d'interprétation associé au module d'adaptation (`decoder_url`). Ce nœud peut également être nommé grâce à l'attribut `DEF` de façon à pouvoir y faire référence ailleurs dans le fichier conteneur.

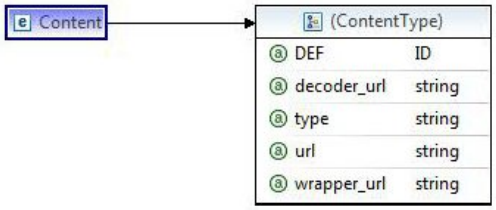


FIGURE *7.3 – Le nœud Content de 3DFC.*

La table 7.1 donne un exemple simple de fichier 3DFC qui encapsule deux fichiers, l'un au format t_1 et l'autre au format t_2. Dans cet exemple, nous n'avons pas renseigné les attributs qui donnent les adresses des modules d'adaptation et des modules d'interprétation. Dans ce cas, le système se charge de trouver le couple module d'adaptation-module d'interprétation approprié. Ce mécanisme sera explicité dans la suite de ce chapitre.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <SGA_3DContainer>
4    <Group DEF="MyGroupNode">
5          <Content DEF="File_1" type="t1" decoder_url="" wrapper_url="" url="
               file1.t1" />
6          <Transform DEF='TRANS' translation='-80 60 50' scale="2 2 2" >
7             <Content DEF="File_2" type="t2" decoder_url="" wrapper_url="" url="
                  file2.t2" />
8          </Transform>
9    </Group>
10 </SGA_3DContainer>
```

TABLE *7.1 – Exemple d'un fichier 3DFC.*

7.2.1.2 Les nœuds d'interaction

Notre modèle de format conteneur prévoit trois nœuds d'interaction :
- un nœud `Route`,
- un nœud `Match`,
- un nœud `Converter`.

Le nœud `Route` présenté par la figure 7.4 rend possible la connexion de deux propriétés équivalentes de deux nœuds issus de fichiers différents. Il fonctionne sur le même principe que le nœud `Route` du format X3D avec un nœud émetteur issu d'un premier fichier et un nœud récepteur issu d'un second fichier. Par propriétés équivalentes nous désignons des propriétés de nœuds qui ont le même type. La propriété `translation` d'un nœud `Transform` de X3D et la propriété `translate` d'un nœud `node` de Collada sont par exemple équivalentes car elles s'expriment toutes deux à l'aide d'un vecteur.

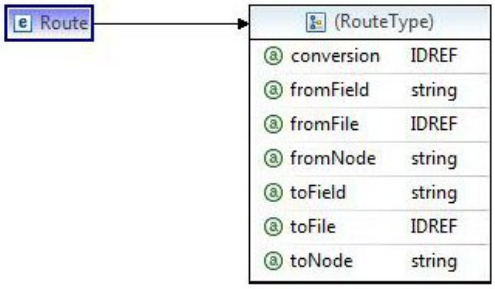


FIGURE 7.4 – *Le nœud `route`.*

Notre nœud `Route` a sept attributs :
- `fromFile` : l'identifiant,dans le fichier 3DFC, du fichier émetteur défini dans un attribut `DEF`,
- `toFile` : l'identifiant, dans le fichier 3DFC, du fichier récepteur défini dans un attribut `DEF`,
- `fromNode` : l'identifiant du nœud émetteur dans son graphe de scène de format,
- `toNode` : l'identifiant du nœud récepteur dans son graphe de scène de format,
- `fromField` : le nom de la propriété du nœud `fromNode` à connecter en émission,
- `toField` : le nom de la propriété du nœud `toNode` à connecter en réception,
- `converter` : l'indentifiant d'un nœud `Converter` du fichier 3DFC. Cet attribut est optionnel, il permet si besoin de convertir dans une autre unité, la valeur de la propriété du nœud émetteur. En effet, même si les propriétés connectées ont le même type, elles ne sont pas forcément exprimées dans la même unité.

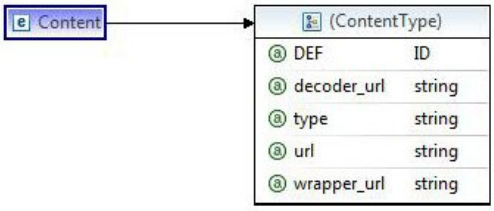


FIGURE 7.5 – *Le nœud `converter`.*

Le nœud `Converter` sert à définir une conversion. Comme l'illustre la figure 7.5, il a deux attributs. Un attribut `DEF` pour lui attribuer un identifiant à l'intérieur du fichier 3DFC et un attribut `type` qui donne le type de la conversion. La valeur de l'attribut type est une chaîne de caractère qui décrit la conversion. Elle permet de définir aussi bien un changement de repère qu'une conversion de valeur comme le passage d'un angle d'Euler à un quaternion pour l'expression

d'une rotation par exemple ou encore la conversion d'une valeur de couleur ou de transparence qui varie beaucoup d'un format à un autre.

Le nœud `Match` permet d'extraire une propriété d'un nœud dans un fichier émetteur et de l'attribuer à un autre nœud d'un fichier récepteur. Nous pouvons ainsi attribuer à un nœud d'un graphe de scène de format donné, une propriété qu'il ne possède pas et même une propriété que n'offre pas son format d'origine.

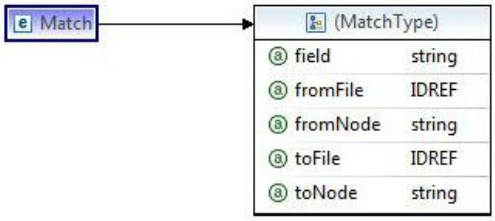


FIGURE *7.6 – Le nœud* `match`.

Le nœud `Match` a cinq attributs comme le montre la figure 7.6 :

- `fromFile` : l'identifiant,dans le fichier 3DFC, du fichier émetteur définit dans un attribut `DEF`,
- `toFile` : l'identifiant, dans le fichier 3DFC, du fichier récepteur définit dans un attribut `DEF`,
- `fromNode` : l'identifiant du nœud émetteur dans son graphe de scène de format,
- `toNode` : l'identifiant du nœud récepteur dans son graphe de scène de format,
- `field` : le nom de la propriété à attribuer dans son graphe de scène de format.

La table 7.2 nous donne un exemple de fichier 3DFC avec des nœuds d'interaction. Ce fichier encapsule deux fichiers, `File_1` et `File_2`. Un nœud `Route` (ligne 11) crée un chemin entre la valeur de la propriété `property_2` d'un nœud du fichier `File_2` et la valeur de la propriété `property_1` d'un nœud du fichier `File_1`. Les deux graphes de scène n'utilisant pas les mêmes unités, un nœud `Converter` (ligne 12) nommé `converter_1` définit le type de conversion à effectuer. On trouve également à la ligne 10 un nœud `Match` qui attribue la propriété `property` d'un nœud du fichier `File_1` à un nœud du fichier `File_2`. Notons que les nœuds et les propriétés des nœuds liés par des nœuds `Route` ou des nœuds `Match` pourraient avoir le même nom, 3DFC ferait la distinction car ils sont issus de fichiers différents.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <SGA_3DContainer>
4    <Transform DEF="TransNode_1" translation="-10 0 0" rotation="0 0 1 15">
5      <Content DEF="File_1" type="t1" decoder_url="" wrapper_url="" url="file1.t1
         "/>
6      <Content DEF="File_2" type="t2" decoder_url="" wrapper_url="" url="file2.t2
         "/>
7    </Transform>
8    <Match fromFile="File_1" fromNode="Node_1" toFile="File_2" toNode="Node_2"
       field="property"/>
9    <Route fromFile="File_2" fromNode="Node_2" fromField="property_2" toFile="
       File_1" toNode="Node_1" toField="property_1" conversion="converter_1" />
10   <Converter DEF="converter_1" type="MaConversion" />
11 </SGA_3DContainer>
```

TABLE *7.2 – Un exemple de fichier 3DFC définissant des interactions entre les fichiers qu'il encapsule.*

7.2.2 Utilisation du 3DFC

Dans cette partie, nous allons expliquer comment le format conteneur est intégré dans une application basée sur le SGA. Nous verrons la façon dont le graphe de scène, défini dans un fichier 3DFC, est interprété lors de son chargement et comment ce graphe de scène est maintenu pendant le déroulement de l'application.

7.2.2.1 Intégration du 3DFC avec le SGA

Notre modèle de conteneur de fichiers 3D se base sur le SGA pour le chargement des fichiers encapsulés. Il s'intègre à notre architecture SGA comme un autre format de fichier ; son intégration se fait par l'intermédiaire d'un module d'interprétation 3DFC et d'un module d'adaptation 3DFC qui communique avec le noyau du SGA. Le module d'interprétation 3DFC est essentiellement constitué d'un analyseur syntaxique pour les fichiers 3DFC et d'un module de conversion. C'est en effet le module d'interprétation qui interprète et opère les conversions définies dans les fichiers 3DFC au travers des nœuds `Converter`. Le schéma 7.7 illustre l'intégration de 3DFC avec le SGA.

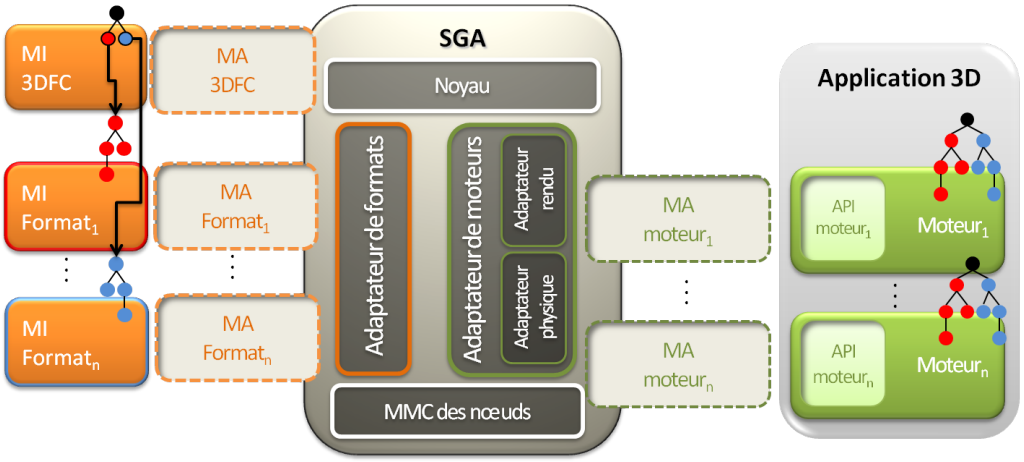


FIGURE 7.7 – *L'intégration de 3DFC au SGA.*

Dans l'état actuel du développement, les fichiers 3DFC sont saisis manuellement. Il est donc nécessaire de connaître le contenus des fichiers encapsulés pour ensuite pouvoir établir des interactions entre ces fichiers. Il faut par exemple connaître les noms des nœuds à mettre en relation mais aussi connaître leurs unités de valeurs afin de préciser une conversion si nécessaire. Il n'y a pas pour l'instant de vérification des types ou des unités des propriétés des nœuds connectées ; si ceux-ci s'avèrent incompatibles alors l'effet de la connexion sur la scène finale est indéfini. Il est cependant envisageable d'automatiser une partie de cette tâche. Il est par exemple possible d'annoter les fichiers connectés avec un modèle d'interaction tel que le modèle MIM proposé par Chmielewski [Chm12]. Nous pourrions également fournir au module d'adaptation 3DFC des fichiers de connexion pour chaque format en se basant sur ce même modèle. Cela permettrait d'offrir une aide à l'établissement de connections sans avoir à modifier les fichiers chargés.

7.2.2.2 Le chargement d'un fichier 3DFC par le SGA

Pour expliquer comment le SGA gère le chargement des fichiers 3DFC, nous allons reprendre l'exemple de la table 7.2. Notre fichier 3DFC encapsule trois fichiers, `File_1` et `File_2`. Il définit également deux connexions entre deux de ces fichiers grâce à un nœud Route et à un nœud Match. Le résultat du chargement de ce fichier par le SGA est illustré par la figure 7.8.

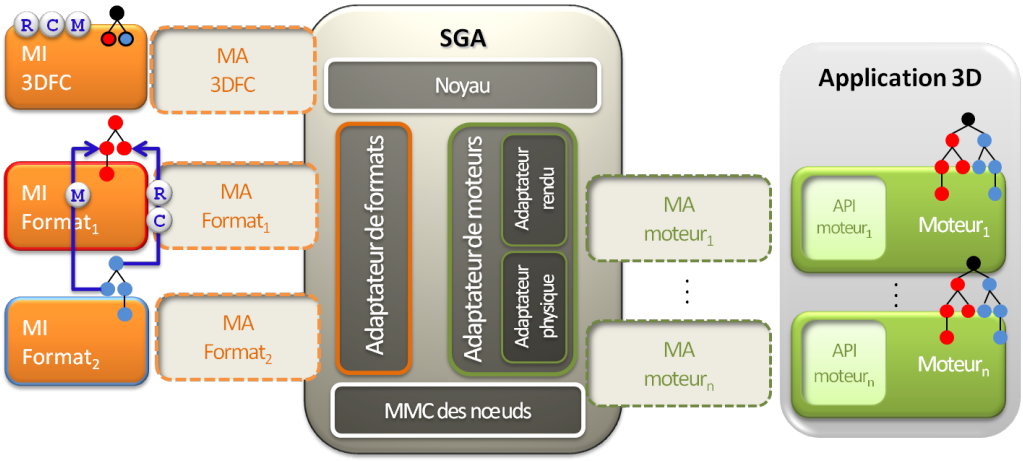


FIGURE *7.8 – Le chargement du fichier de la table 7.2 par le SGA.*

Le fichier 3DFC est chargé par l'application de la même façon que tous les fichiers 3D, comme le montre le schéma 7.9 ; l'application transmet au noyau l'adresse du fichier (message 1) puis le noyau charge le module d'adaptation de format approprié (message 2) qui prend le relais (message 3) et délègue finalement le chargement du fichier à son module d'interprétation (message 4). Le module d'adaptation 3DFC reprend ensuite la main pour procéder à l'adaptation du fichier (messages 5 et 6). Lorsqu'il atteint le nœud `Group` de la ligne 4, il demande l'adaptation de ce nœud dans tous les graphes de scène de moteur par l'intermédiaire du noyau (messages 8 et 9). Il gère ensuite le nœud `Content` de la ligne 5 et demande son chargement au noyau (messages 10 et 11). Le noyau charge alors le module d'adaptation de format approprié (message 12) à qui il délègue l'adaptation du graphe de scène de format qu'il encode (message 13). Le module d'adaptation 3DFC procède de la même façon pour les nœuds de description restants.

La gestion du nœud `Route` de la ligne 11 nécessite plus d'échanges entre les composants du SGA comme le montre la figure 7.11.

```
1 <Route fromFile="File_2" fromNode="Node" fromField="property_2" toFile="File_1"
      toNode="Node_1" toField="property_1" conversion="converter_1" />
2 <Converter DEF="converter_1" type="MaConversion" />
```

Lorsque ce nœud est atteint par le module d'adaptation (message 1), celui-ci commence par demander au module d'adaptation de format du fichier `File_2` l'information de la valeur de l'attribut `property_2` du nœud `Node` par l'intermédiaire du noyau (message 2). Le noyau à l'aide de l'index de nœuds retrouve le module d'adaptation de format du fichier (message 3) et lui demande l'information demandée (message 4) et la retourne au noyau. Le noyau la transmet à son tour au module d'adaptation 3DFC qui procède à sa conversion selon les instructions de conversion données par le nœud `converter_1`. Pour cette opération, le module d'adaptation 3DFC s'appuie sur un module qui associe à chaque chaîne de caractère un calcul de conversion. Une fois la valeur convertie dans la bonne unité, le module d'adaptation 3DFC demande au noyau de mettre à jour la valeur de la propriété `property_1` du nœud `Node_1` dans le graphe de scène du fichier `File_1` (message 6). Le noyau retrouve ensuite le module d'adaptation approprié aidé par l'index de nœuds (message 7) puis propage la mise à jour du nœud à son module d'adaptation de format (message 8). C'est ce dernier qui, une fois son graphe de scène de format mis à jour, propagera cette modification à tous les graphes de scène de moteur via le noyau (message 9).

La gestion du nœud `Match` entraîne également un échange de messages entre les composants du SGA. Celui-ci est illustré dans la figure 7.13.

```
1 <Match fromFile="File_1" fromNode="Node_1" toFile="File_2" toNode="Node_1"
    field="property"/>
```

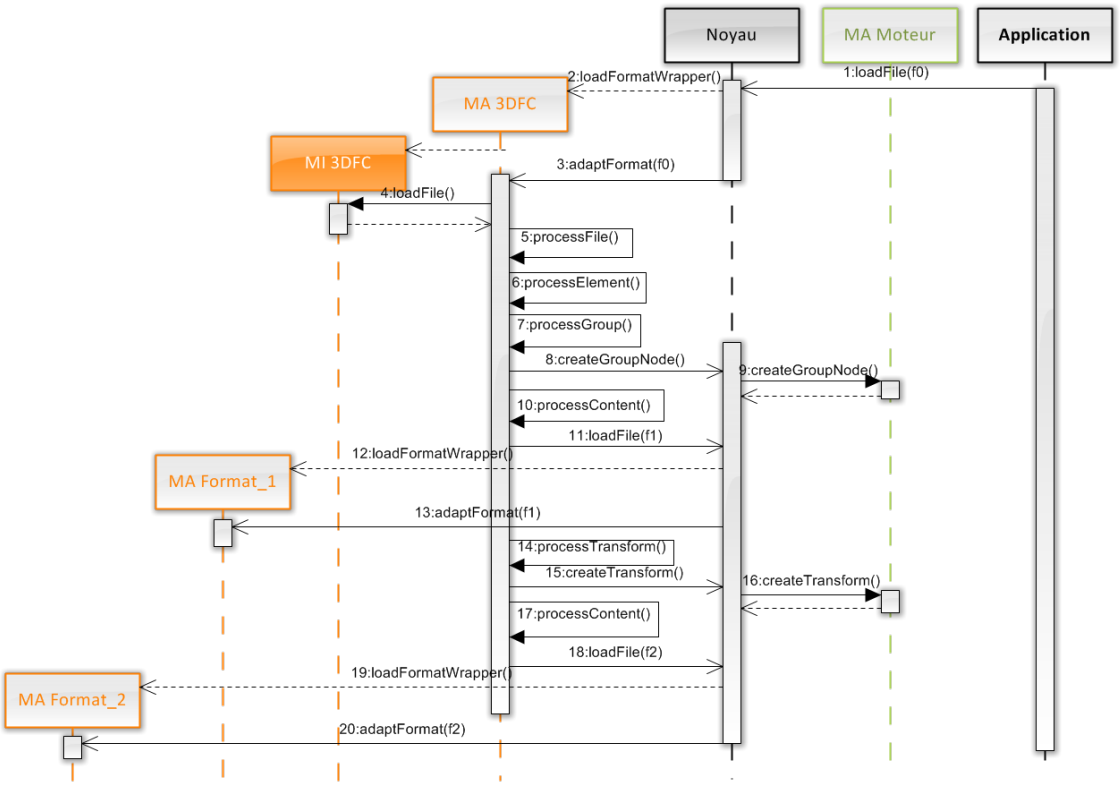


FIGURE 7.9 – *Les messages échangés pour le chargement du fichier de la table 7.2 par le SGA.*

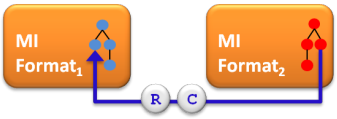


FIGURE 7.10 – *Illustration des nœuds* `Route` *et* `Converter` *du fichier de la table 7.2.*

Lorsque le module d'adaptation 3DFC atteint le nœud `Match` (message 1), il interprète celui-ci et transmet l'instruction au noyau (message 2). Le noyau demande à l'index de nœuds le module d'adaptation du nœud émetteur (message 3) puis lui propage l'instruction (message 4). Le module d'adaptation du format du fichier `File_1` va alors retrouver la valeur de la propriété `property` du nœud `Node_1` (message 5) et demander au noyau de l'attribuer à toutes les adaptations du nœud `Node_1` du fichier `File_2` dans les graphes de scène de moteur (messages 6 et 7). La propriété attribuée au nœud `Node_1` du fichier `File_2` étant non spécifiée dans son format d'origine, c'est le module d'adaptation du format émetteur qui se charge de son interprétation et de son adaptation.

7.2.2.3 Le fonctionnement du modèle de 3DFC pendant l'exécution

Pendant le déroulement de l'application, le module d'adaptation 3DFC se charge de mettre à jour les interactions qu'il définit. La figure 7.14 montre les messages échangés pour mettre à jour les nœuds `Route` définis dans l'exemple de la table 7.2.

A chaque mise à jour du temps par l'application, celle-ci en informe le module d'adaptation 3DFC par l'intermédiaire du noyau (messages 1 et 2). Le module d'adaptation 3DFC va alors pour chaque nœud `Route` qu'il définit, interroger le module d'adaptation de format du nœud émetteur

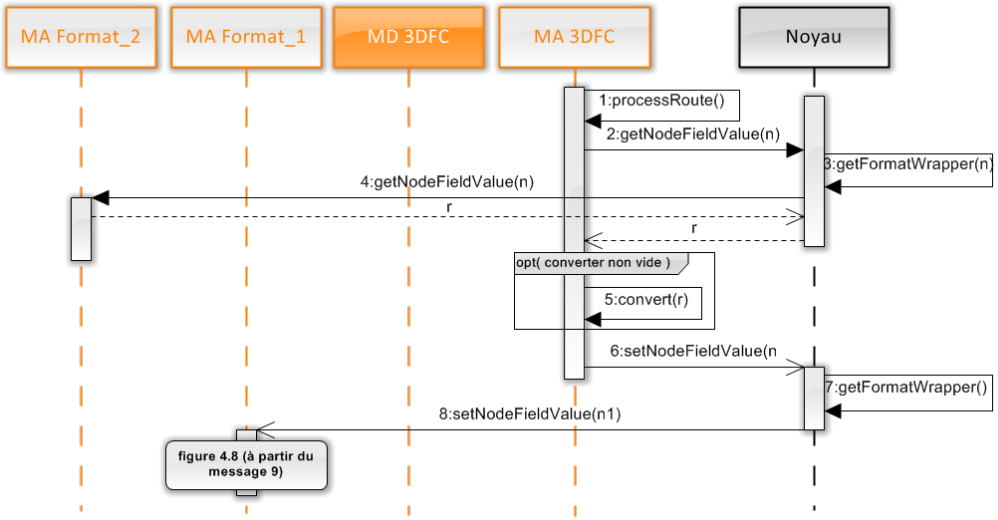


FIGURE *7.11* – *Les messages échangés pour le chargement du nœud* `Route` *du fichier de la table 7.2 par le SGA.*

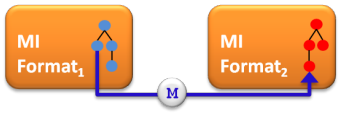


FIGURE *7.12* – *Illustration du nœud* `Match` *du fichier de la table 7.2.*

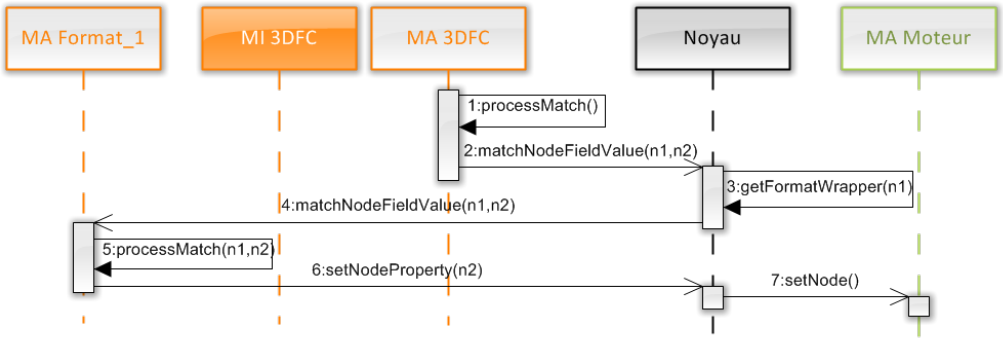


FIGURE *7.13* – *Les messages échangés pour le chargement du nœud* `Match` *du fichier de la table 7.2 par le SGA.*

pour connaître la valeur de sa propriété routée via le noyau (messages 3, 4 et 5). Une fois la valeur de la propriété connue, si l'attribut `conversion` de la Route est défini, le module d'adaptation 3DFC calcule la valeur convertie (message 6). Enfin, il transmet au module d'adaptation de format du nœud récepteur la nouvelle valeur de la propriété par l'intermédiaire du noyau (messages 7, 8 et 9).

Pour les propriétés attribuées à un nœud grâce à un nœud `Match`, elles n'entraînent pas d'échange de message au cours du déroulement de l'application entre le noyau et les modules d'adaptation de format. En effet, la propriété ainsi attribuée n'est prise en compte que par les graphes de scène de moteur. Elle n'a d'existence que dans ces graphes de scène et ne peut donc être modifiée qu'à l'initiative d'un des moteurs.

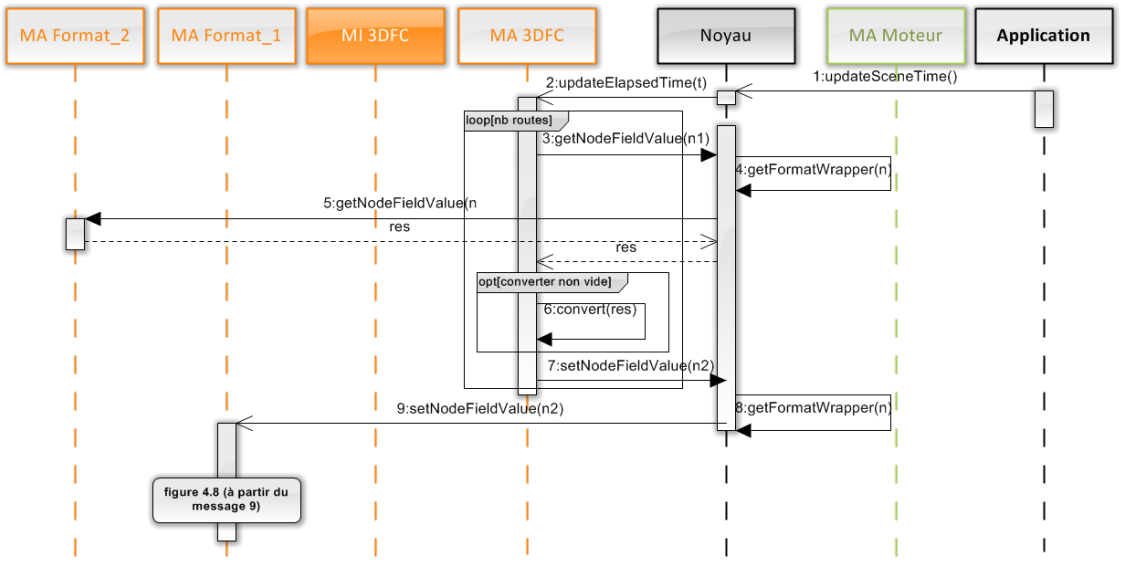


FIGURE 7.14 – *Les messages échangés pour la mise à jour du nœud* `Route` *du fichier de la table 7.2 par le SGA pendant le déroulement de l'application.*

7.3 INSTANCIATION DU MODÈLE 3DFC ET INTÉGRATION À NOTRE IMPLÉMENTATION DU SGA

Nous avons réalisé une instanciation d'un module d'adaptation 3DFC. Celui-ci s'appuie sur un décodeur que nous avons développé et qui se base sur l'analyseur syntaxique TinyXML [2]. Nous avons ensuite intégré ce module d'interprétation et ce module d'adaptation à notre implémentation du SGA. Ce système nous permet de créer des fichiers 3DFC qui référencent des fichiers Collada et des fichiers X3D, de les charger dans notre application et de les faire interagir dans la fenêtre de rendu comme le résume la figure 7.15.

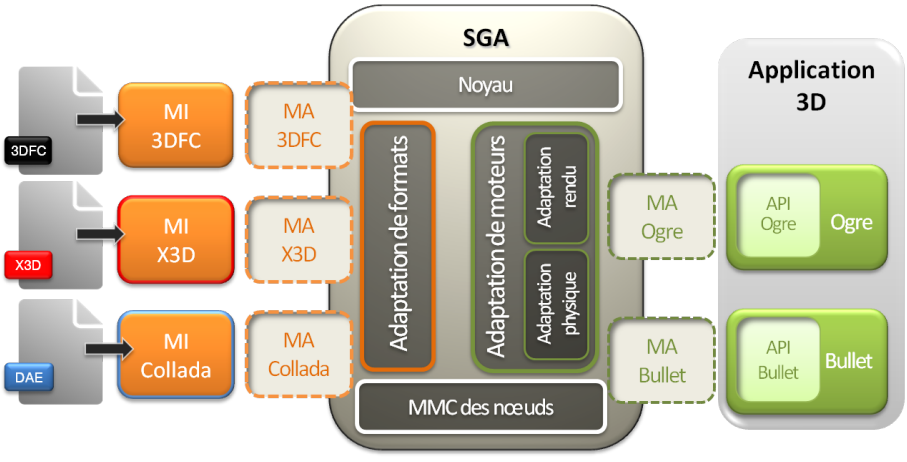


FIGURE 7.15 – *Notre instanciation du modèle de 3DFC et son intégration à notre implémentation du SGA.*

Dans la suite de ce paragraphe, nous allons développer un exemple qui illustre l'interopéra-

2. http ://www.grinninglizard.com/tinyxml/

bilité offerte par notre conteneur de fichiers 3D, au travers d'une scène composée à partir d'un fichier X3D et d'un fichier Collada.

7.3.1 Description de scène composée

L'exemple ci-dessous (cf figure 7.16) montre comment composer une scène à partir de fichiers X3D et Collada en utilisant un fichier conteneur 3DFC. Le fichier conteneur permet à partir d'un seul fichier, de charger plusieurs modèles aux formats divers, de les positionner dans la scène finale et de les redimensionner afin de les mettre à l'échelle sans qu'il soit nécessaire de modifier les fichiers impliqués.

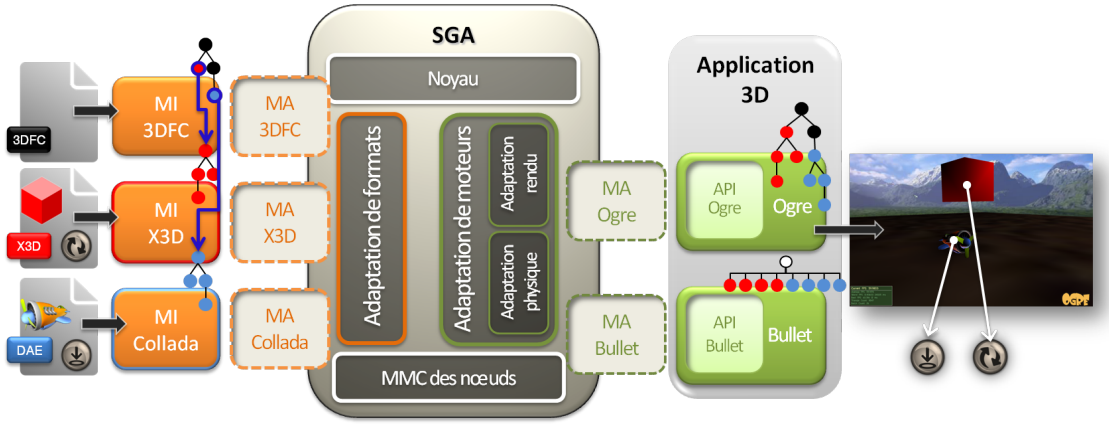


FIGURE 7.16 – *Illustration de l'exemple 7.1*

Exemple 7.1 ***En entrée :*** *Nous avons dans cette exemple trois fichiers qui sont utilisés : un fichier 3DFC qui référence un fichier X3D et un fichier Collada :*

- *le fichier X3D : il représente un cube rouge dans un monde constitué d'une voûte céleste et d'un sol. Le fichier X3D définit de plus une interaction et un comportement : lorsque l'utilisateur clique sur le cube, ce dernier se met à tourner sur lui-même autour de l'axe Y.*
- *le fichier Collada : il représente un avion, de plus le fichier Collada définit des propriétés physiques pour le monde et l'avion.*
- *un fichier 3DFC : ce fichier encapsule les deux fichiers précédents (cf table 7.3). On trouve tout d'abord le fichier X3D (ligne 6) puis le fichier Collada encapsulé dans un nœud* `Transform` *qui permet de placer les modèles X3D et Collada côte à côte et de redimensionner le modèle Collada afin de le mettre à l'échelle du modèle X3D.*

En sortie : *Nous pouvons visualiser les deux modèles, le cube et l'avion, dans la fenêtre de rendu Ogre. Le cube tourne sur lui-même lorsque nous cliquons dessus et l'avion tombe sur le sol inclut dans le modèle X3D.*

Fonctionnement : *Le fichier 3DFC référence les fichiers X3D et Collada, il permet d'utiliser ces deux fichiers sans les modifier ni les charger explicitement dans l'application. Le chargement des ces fichiers se fait comme expliqué dans le diagramme 7.9. La prise en compte des propriétés de chacun des fichiers se fait au sein du module d'interprétation dédié à chaque format comme pour un chargement séparé des fichiers.*

7.3.2 Mixer les fonctionnalités des formats 3D

Le format conteneur 3DFC nous permet de mixer facilement les fonctionnalités de plusieurs formats sans modifier les fichiers originaux. Les deux exemples suivants montrent comment mixer les fonctionnalités de X3D et Collada afin d'enrichir la scène finale.

```
<?xml version="1.0" encoding="UTF-8"?>

<SGA_3DContainer>
   <Group DEF="MyGroupNode">
      <Content DEF="X3DFile" type="x3d" decoder_url="" wrapper_url="" url="../
          x3d/cube.x3d" />
      <Transform DEF='TRANS' translation='-80 60 50' scale="2 2 2" >
         <Content DEF="DAEFile" type="dae" decoder_url="" wrapper_url="" url="
             ../collada/Seymour_Plane/seymourplane.dae" />
      </Transform>
   </Group>
</SGA_3DContainer>
```

TABLE 7.3 – *Ce fichier 3DFC crée une scène composée d'un fichier X3D et d'un fichier Collada ; il est utilisé dans les exemples 7.1, 7.2 et 7.3.*

7.3.2.1 Utilisation du nœud `Route`

L'exemple suivant (cf figure 7.17) montre comment utiliser le nœud Route de 3DFC afin de tirer partie des animations définies dans un fichier pour ensuite les calquer sur un modèle d'un autre fichier.

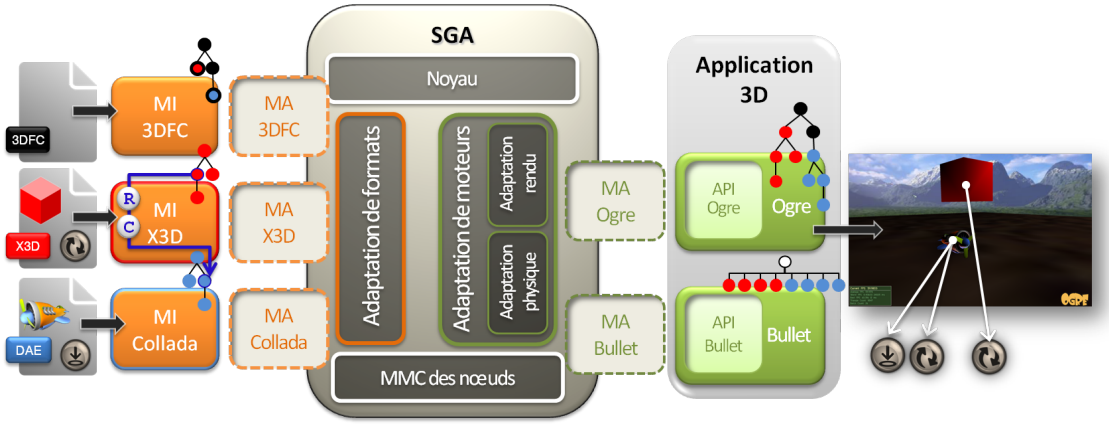


FIGURE 7.17 – *Illustration de l'exemple 7.2*

Exemple 7.2 ***En entrée :*** *Nous reprenons le fichier 3DFC de l'exemple précédent (cf 7.3) auquel nous ajoutons un nœud* `Route` *et un nœud* `Converter` *:*

```
   <Route fromFile="X3DFile" fromNode="AnimationOI" fromField="value_changed"
       toFile="DAEFile" toNode="prop" toField="rotate" conversion="convertor_1"
        />
   <Converter DEF="convertor_1" type="VectRotToQuaternion" />
```

En sortie : *Dans la fenêtre de rendu, lorsque l'utilisateur clique sur le cube rouge, nous pouvons voir le cube et l'hélice de l'avion tourner simultanément.*

Fonctionnement : *Pour calquer l'animation du cube sur l'hélice de l'avion, deux lignes ont été ajoutées dans le fichier 3DFC mais les fichiers X3D et Collada n'ont pas été modifiés. La première ligne définit un nœud* `Route` *qui lie la valeur du champ* `value_changed` *du nœud d'interpolation nommé* `AnimationOI` *du fichier X3D à la valeur du champ* `rotate` *du nœud décrivant l'hélice de l'avion nommé* `prop` *dans le fichier Collada. La seconde ligne définit une conversion qui calcule une valeur de rotation exprimée à l'aide*

d'un vecteur et d'un angle de rotation en une valeur exprimée à l'aide d'un quaternion. X3D et Collada n'utilisent pas la même unité pour exprimer une rotation, le nœud Converter calcule la conversion avant que la valeur de la rotation ne soit envoyée au module d'adaptation Collada comme le montre la figure 7.14.

7.3.2.2 Utilisation du nœud `Match`

L'exemple suivant (cf figure 7.18) montre comment, à l'aide du nœud `Match` de 3DFC, nous pouvons attribuer une propriété issue d'un premier format à un nœud défini dans un autre format afin d'enrichir la scène finale.

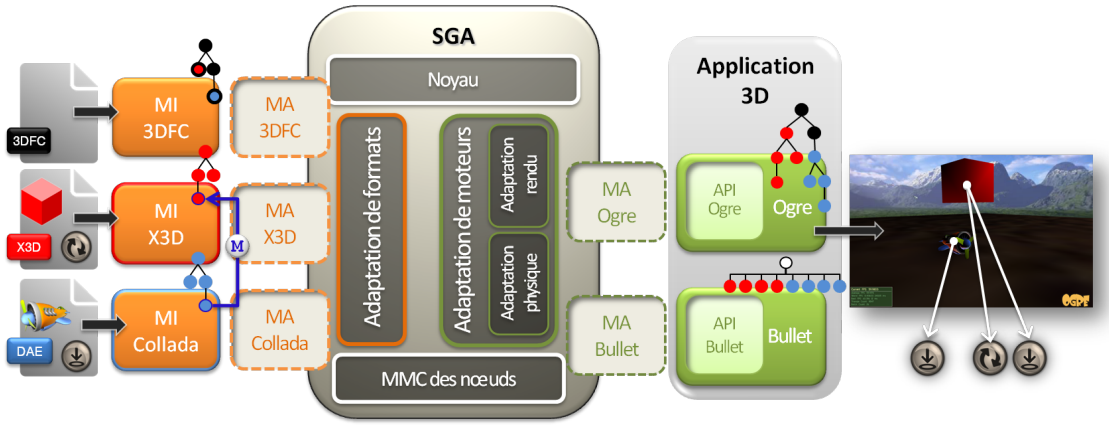


FIGURE *7.18 – Illustration de l'exemple 7.3*

Exemple 7.3 ***En entrée :*** *Dans cette exemple, nous reprenons le fichier 3DFC donné par la table 7.3 qui crée une scène composée d'un cube rouge X3D et d'un avion Collada. Nous ajoutons à ce fichier deux lignes :*

```
1    <Match fromFile="DAEFile" fromNode="plane" toFile="X3DFile" toNode="cube_RED
         " field="mass"/>
2    <Match fromFile="DAEFile" fromNode="plane" toFile="X3DFile" toNode="cube_RED
         " field="dynamic"/>
```

En sortie : *Nous pouvons voir dans la fenêtre de rendu, le cube X3D et l'avion Collada tomber tous les deux sur le sol définit dans le fichier X3D.*

Fonctionnement : *Les deux lignes ajoutées au fichier 3DFC de départ ont permis d'attribuer la masse et la valeur du champ `dynamic` de l'avion du fichier Collada au cube du fichier X3D. Ainsi, dans le graphe de scène de Bullet, le cube X3D possède une masse et est considéré comme un objet dynamique, il est donc pris en compte dans la simulation générée par le moteur physique. En conséquence, à chaque pas de simulation, la position du cube dans la scène est mise à jour comme illustré dans le diagramme de séquences 6.19. L'ajout de ces deux lignes a permis d'attribuer une propriété à un nœud initialement défini dans un format qui n'offre pas cette propriété.*

Ces deux exemples montrent la façon dont peuvent être utilisés les trois nœuds d'interaction introduit par le modèle 3DFC. Ils peuvent bien entendu être combinés dans un même fichier auquel cas nous obtiendrons une scène dans laquelle les deux objets tombent et tournent sur eux-mêmes lorsqu'on clique sur le cube. A travers ces exemples, nous voyons comment tirer partie des fonctionnalités propres à chaque format afin de les combiner entre eux et obtenir de manière aisée des scènes plus riches.

7.4 SYNTHÈSE ET CONCLUSION

7.4.1 SYNTHÈSE

L'utilisation d'un conteneur de fichier 3D offre de nombreux avantages. Tout d'abord, un conteneur permet de composer des scènes sans contraintes de compatibilité de format mais, surtout, il permet de faire communiquer et interagir leurs contenus. Ces capacités reposent sur trois nœuds de description et trois nœuds d'interaction qui reprennent des concepts simples et déjà utilisés dans le domaine de l'informatique graphique. Les nœuds de description sont en effet similaires à ceux de la majorité des formats 3D. Quant aux nœuds d'interaction, leurs spécifications sont facilement appréhendables.

Le nœud `Route` est utilisé pour copier une propriété d'un nœud donné à un nœud d'un autre graphe de scène de format. La propriété copiée est partagée entre deux fichiers potentiellement encodés dans des formats différents. Le nœud `Route` permet d'organiser et de découper une scène en plusieurs fichiers de façon à améliorer la réutilisabilité de ceux-ci dans d'autres scènes. De plus, cela permet d'animer facilement un nœud issu d'un format qui ne prévoit pas cette fonctionnalité. Pour cela, nous avons deux options :

- créer une Route entre un nœud animé et ce nœud,
- créer une Route entre un nœud d'animation et ce nœud.

Enfin, grâce au nœud `Converter`, nous avons la possibilité de créer des liens à l'aide de nœuds `Route` entre des propriétés de nœud sans être bloqué par des problèmes de compatibilité d'unité.

Le nœud `Match` est quant à lui utilisé pour attribuer une propriété d'un nœud N_1 à un nœud N_2 issu d'un format ne proposant pas cet attribut. Grâce au nœud `Match`, nous sommes capables d'ajouter une propriété à un modèle quelque soit le format dans lequel est défini ce nœud sans modifier le fichier original. La propriété ainsi attribuée est définie par le format du nœud émetteur et est interprétée ensuite dans le SGA par le module d'interprétation de ce format.

La simplicité du modèle du 3DFC et sa sobriété visent à faciliter son intégration dans un format existant. En effet, cela nécessiterait de n'ajouter à ce format que les trois nœuds d'interaction ; les nœuds de description étant remplacés par leurs équivalents dans le format en question.

7.4.2 CONCLUSION

L'utilisation combinée d'un conteneur de formats 3D et du système du SGA nous permet de tirer partie des propriétés de chaque format 3D, de les combiner à d'autres fonctionnalités issues d'autres formats afin d'obtenir des scènes tridimensionnelles enrichies. Cela sans qu'il soit nécessaire de modifier au préalable les fichiers impliqués. Par ailleurs, la solution du 3DFC concorde avec les éléments fixés par notre cadre d'étude, à l'exception de l'indépendance avec les composants de rendu mais elle n'est pas concernée par cette caractéristique. Elle tient compte également des caractéristiques que nous avons établies concernant les contenus. Le tableau 2.4.1 reprend l'ensemble de ces éléments et les confronte au modèle du 3DFC.

	Caractéristiques de la solution	**3DFC**
Cadre initial	utilisation directe des contenus	-
	utilisation directe des composants logiciels	-
	communication entre contenus et composants	-
	communication inter-contenus	✓
	communication inter-composants	-
Contenus	préserve les fonctionnalités du format des fichiers chargés	✓
	pas de conversion	✓
	préserve les modèles encodés dans les fichiers chargés	✓

TABLE *7.4 – Le modèle de conteneur de fichier 3DFC et les caractéristiques d'une solution d'interopérabilité satisfaisante.*

CONCLUSION

LES environnements virtuels 3D sont aujourd'hui utilisés dans de nombreux domaines et couvrent de multiples usages qui vont du divertissement à l'apprentissage en passant par la conception pour l'industrie. Si leur nombre ne cesse de croître chaque année, un problème récurrent freine leur adoption et leur diffusion. Il est en effet aujourd'hui très difficile d'importer des données dans un environnement virtuel autre que celui pour lequel ces données ont été créées. Ceci entraîne de nombreux autres problèmes qui ralentissent la création d'environnements virtuels 3D, augmentent leur coût de production et limitent leur accès aux utilisateurs.

Nous présentons ici une solution qui se propose de réconcilier les modèles existants et à venir, de façon à rendre interopérables les contenus et les composants logiciels des environnements virtuels 3D. Cette solution repose sur deux éléments complémentaires : une architecture logicielle et un modèle de conteneur de formats 3D.

Nous avons appelé notre système l'adaptateur de graphes de scène ou SGA. Il s'agit d'une architecture générique et modulaire qui permet le chargement de plusieurs formats 3D dans la plupart des composants logiciels des environnements virtuels 3D. La conception du SGA se base sur des observations de similitudes des formats et des composants logiciels. Ils partagent en effet une structuration en graphe de scène. Le SGA a non seulement pour rôle d'adapter tous les graphes de scène encodés dans les fichiers 3D en graphes de scène pour les composants qui sont utilisés par l'environnement virtuel mais également de gérer la synchronisation de ces graphes de scène au cours du déroulement de l'application de l'environnement virtuel. Nous avons réalisé une implémentation de cette architecture ainsi que plusieurs instanciations qui permettent le chargement de fichiers au format X3D et COLLADA dans une application dont le rendu se base sur le moteur de rendu graphique Ogre3D et le moteur physique Bullet. Ceci nous a permis de démontrer la faisabilité de cette solution et d'évaluer ses performances.

Notre modèle de conteneur de formats 3D permet non seulement de composer des scènes faites à partir de plusieurs fichiers 3D mais aussi de combiner leurs fonctionnalités et de les faire interagir dans l'environnement virtuel. Ce modèle est appelé 3DFC pour conteneur de fichiers 3D et il repose sur le système du SGA pour l'adaptation des graphes de scène encapsulés dans les fichiers référencés par le conteneur. Nous avons réalisé une instanciation pour le modèle 3DFC que nous avons intégrée à notre implémentation du SGA. Nous avons ainsi pu mixer dans une même scène des modèles X3D et COLLADA et combiner les fonctionnalités d'interactions offertes par X3D avec les propriétés physiques autorisées dans COLLADA.

CONTRIBUTION

Lors des premières phases de conception et d'implémentation, une version initiale de l'architecture du SGA a été publiée sous forme d'un poster à ICAT 2010 [BBRDA10]. Nous y présentons la conception du modèle de réconciliation entre les contenus et les composants logiciels des environnements virtuels 3D ainsi que l'architecture modulaire pour la communication entre des

graphes de scène de formats et des graphes de scène de moteurs basée sur un modèle de réconciliation. L'architecture définitive a ensuite été présentée lors de la conférences Web3D 2010 [BRDA11]. Nous y présentons l'implémentation de notre architecture ainsi que la réalisation de nos deux instanciations de modules d'adaptation de formats et de nos deux instanciations de modules d'adaptation de moteurs. Nous y introduisons également l'interface de programmation de réconciliation pour les graphes de scène. La composition de scènes décrites dans des formats différents a été détaillée dans un article paru dans la revue JVWR (Journal of Virtual World Research)[BBDRA11]. Enfin, la mise en correspondance de noeuds décrits dans des formats différents pour les faire interagir grâce au 3DFC a été présentée lors de la conférence Web3D 2012 [BBRDA12]. Nous y avons présenté la conception de notre modèle de conteneur de formats 3D et la réalisation d'un module d'interprétation et d'un module d'adaptation pour 3DFC.

RÉSULTATS

Face à la prolifération des formats 3D et à la difficulté d'établir un standard unique dans le domaine de la réalité virtuelle, nous pensons qu'une solution d'interopérabilité cherchant à réconcilier les modèles existants et à venir est l'alternative la plus avisée. Les solutions du SGA et du 3DFC entrent dans cette catégorie car elles permettent une interopérabilité entre les formats qui préserve leurs contenus et leurs propriétés. Ces solutions rendent également possible la communication entre les formats eux-mêmes, entre les composants logiciels des environnements virtuels 3D mais aussi entre les formats et les composants logiciels. Cette faculté permet pour une application d'un environnement virtuel reposant sur le SGA de synchroniser durant toute la durée de son exécution les graphes de scène des formats et ceux des composants logiciels. Ainsi, il nous est possible de combiner n'importe quel format 3D avec n'importe quel environnement virtuel 3D quels que soient les composants logiciels utilisés par ce dernier. Nous avons répondu à la problématique soulevée par ces travaux tout en respectant le cadre initial que nous nous étions fixé (cf tableau 7.4.2). Nous avons également tenu compte des conclusions tirées de l'étude des solutions existantes pour tenter d'apporter une réponse qui propose un compromis satisfaisant pour une majorité des domaines intéressés par la réalité virtuelle.

	Caractéristiques de la solution	**SGA + 3DFC**
Cadre initial	utilisation directe des contenus	✓
	utilisation directe des composants logiciels	✓
	communication entre contenus et composants	✓
	communication inter-contenus	✓
	communication inter-composants	✓
Contenus	préserve les fonctionnalités du format des fichiers chargés	✓
	pas de conversion	✓
	préserve les modèles encodés dans les fichiers chargés	✓
Composants de rendu	collaboration possible entre eux	✓
Environnements 3D	pas de modification préalable des contenus	✓
	pas de mise en place de protocole	✓

TABLE *7.5 – La combinaison du système du SGA et du modèle de conteneur de fichier 3DFC et les caractéristiques d'une solution d'interopérabilité satisfaisante.*

Le système du SGA a été conçu pour être flexible et extensible, il fournit une interface de programmation qui facilite et guide l'instanciation de nouveaux formats ou composants logiciels qui pourront ensuite être pris en charge par n'importe quelle application d'environnement virtuel reposant sur le SGA. Le SGA ne stocke pas de représentation intermédiaire des graphes de scène mis en relation mais gère une table de mise en correspondances des nœuds de ces graphes de scène. La modification d'un nœud, au cours de l'exécution de l'application, entraîne systémati-

quement une recherche dans cette table ce qui alourdit le processus de rendu synchronisé par le SGA par rapport à un processus normal. Cependant l'impact de cette recherche est faible comparé aux différents processus générés pour le rendu.

PERSPECTIVES

Ces travaux, en répondant à la problématique posée, ont soulevé plusieurs autres questionnements et ouvert des pistes de réflexion. Le modèle d'interopérabilité du SGA, comme de nombreux autres modèles d'interopérabilité, gagnerait en efficacité par son adoption par une communauté. En effet, la réutilisabilité des modules d'adaptation du SGA, permet leur diffusion et leur mise à disposition pour les besoins d'autres applications. Nous avons prévu la mise en place d'un serveur stockant les modules d'adaptation disponibles et le téléchargement direct des modules d'adaptation nécessaires à une application. De même, le modèle du 3DFC a été conçu pour être intégré à un format existant afin de le rendre interopérable avec le plus grand nombre de formats possible.

Concernant le SGA, outre les améliorations et optimisations possibles de notre implémentation, nous aurions aimé proposer d'autres instanciations de modules d'adaptation de formats et de modules d'adaptation de moteurs. Premièrement pour les formats, il aurait été intéressant de tester la possibilité d'utiliser des formats qui ne soient pas basés sur un graphe de scène comme le format Object3D ou encore des formats proposant d'autres fonctionnalités que celles offertes par X3D et Collada. Par exemple des formats qui proposent une subdivision spatiale dans leur graphe de scène. Deuxièmement pour les moteurs, nous aurions aimé tester la possibilité d'utiliser d'autres types de moteurs mais aussi évaluer la possibilité d'utiliser simultanément deux moteurs de même type ; deux moteurs de rendu graphiques par exemple. La généricité de notre interface de programmation rend en effet possible le couplage d'une multitude de moteurs. Il serait même possible d'utiliser des moteurs dont l'interface de programmation n'est pas codée dans le même langage que notre interface de programmation. Pour cela, il nous faudrait remplacer l'interface de programmation par un protocole qui proposerait un échange de messages similaires aux appels de méthode mis en place dans notre implémentation. Une autre évolution intéressante serait l'intégration de l'utilisation d'un module de mise en réseau afin d'évaluer la possibilité d'utiliser notre architecture avec des plateformes d'environnements virtuels en ligne.

Concernant notre modèle de conteneur de fichiers 3D, nous avons prévu de l'enrichir en proposant un nouveau nœud qui permettrait d'ajouter des propriétés à un nœud d'un graphe de scène encapsulé dans le conteneur. Nous pensons notamment attribuer des propriétés d'interaction (sélection, manipulation et relâchement) à un nœud mais aussi attribuer des droits sur les nœuds notamment pour du travail collaboratif.

Il pourrait également être intéressant de fournir à notre module d'adaptation 3DFC un fichier de description pour chaque format. Ce fichier contiendrait par exemple pour chaque type de nœud du format sa sémantique, l'ensemble de ses attributs, leur sémantique ainsi que l'unité et le type de leurs valeurs. Ceci permettrait de mettre en place une aide à la création de connexions entre nœuds issus de fichiers différents en proposant les attributs partageant la même sémantique, le même type et en indiquant si une conversion est nécessaire.

GLOSSAIRE

SGA : Scene Graph Adapter
MA : Module d'Adaptation
MI : Module d'Interprétation
MMC : Module de Mise en Correspondance
3DFC : 3D File Container

BIBLIOGRAPHIE

[3D 09] 3D TECHNOLOGIES R&D : 3DMLW. 2009. http ://www.3dmlw.com.

[AB06] R. ARNAUD et M.C. BARNES : Collada ; sailing the gulf of 3d digital content creation. *Recherche*, 67:02, 2006.

[Bar78] R. BARTLE : Welcome to the home of MUD1 - british legends ! http ://www.british-legends.com/, 1978.

[Bar04] Richard A. BARTLE : *Designing virtual worlds*. New Riders, 2004.

[BBBL01] J.N. BAILENSON, J. BLASCOVICH, A.C. BEALL et J.M. LOOMIS : Equilibrium theory revisited : Mutual gaze and personal space in virtual environments. *Presence : Teleoperators & Virtual Environments*, 10(6):583–598, 2001.

[BBDRA11] Rozenn BOUVILLE BERTHELOT, Thierry DUVAL, Jérôme ROYAN et Bruno ARNALDI : Improving reusability of assets for virtual worlds while preserving 3d formats features. *Journal of Virtual Worlds Research*, 4(3), 2011.

[BBRDA10] Rozenn BOUVILLE BERTHELOT, Jérôme ROYAN, Thierry DUVAL et Bruno ARNALDI : Late Breaking Results : Enabling interoperability between 3D formats through a generic architecture. *In ICAT 2010 (20th International Conference on Artificial Reality and Telexistence)*, Adelaide, Australia, décembre 2010.

[BBRDA12] Rozenn BOUVILLE BERTHELOT, Jérôme ROYAN, Thierry DUVAL et Bruno ARNALDI : *3DFC : a new Container model for 3D File formats compositing*, pages 27–35. Web3D 12. ACM, 2012.

[BC08] Mike BAILEY et Steve CUNNINGHAM : Introduction to computer graphics. *In ACM SIGGRAPH ASIA 2008 courses*, pages 1–103, Singapore, 2008. ACM.

[BDL10] Joshua BELL, Morgaine DINOVA et David LEVINE : VWRAP for virtual worlds interoperability. *IEEE Internet Computing*, 14(1):73–77, 2010.

[Bei97] E. BEIER : A generic approach to computer graphics. *Visualization and mathematics : experiments, simulations, and environments*, page 227, 1997.

[BEJZ09] J. BEHR, P. ESCHLER, Y. JUNG et M. ZÖLLNER : X3dom : a dom-based html5/x3d integration model. *In Proceedings of the 14th International Conference on 3D Web Technology*, pages 127–135, 2009.

[BJK+10] J. BEHR, Y. JUNG, J. KEIL, T. DREVENSEK, M. ZOELLNER, P. ESCHLER et D. FELLNER : A scalable architecture for the html5/x3d integration model x3dom. *In Proceedings of the 15th International Conference on Web 3D Technology*, pages 185–194, 2010.

[BRDA11] Rozenn Bouville BERTHELOT, Jérôme ROYAN, Thierry DUVAL et Bruno ARNALDI : Scene graph adapter : an efficient architecture to improve interoperability between 3d formats and 3d applications engines. *In Proceedings of the 16th International Conference on 3D Web Technology*, Web3D '11, pages 21–29, New York, NY, USA, 2011. ACM.

[BSPM09] Vincent BONNEAU, Tiffany SAUQUET, Sophie PERNET et Laurent MICHAUD : *Web 3D - Du monde virtuel au web immersif*. Numéro M91409. Déc 2009.

[BVOGM07] I.M. BILASCO, M. VILLANOVA-OLIVER, J. GENSEL et H. MARTIN : Semantic-based rules for 3d scene adaptation. *In Proceedings of the twelfth international conference on 3D web technology*, pages 97–100. ACM, 2007.

[CB97] R. CARREY et G. BELL : The annotated vrml97 reference manual. http ://www.cs.vu.nl/ eliens/documents/vrml/reference/BOOK.HTM, 1997.

[Chm12] Jacek CHMIELEWSKI : Describing interactivity of 3d content. *In* Wojciech CELLARY et Krzysztof WALCZAK, éditeurs : *Interactive 3D Multimedia Content*, pages 195–221. Springer London, 2012.

[CNSD+92] Carolina CRUZ-NEIRA, Daniel J. SANDIN, Thomas A. DEFANTI, Robert V. KENYON et John C. HART : The cave : audio visual experience automatic virtual environment. *Commun. ACM*, 35(6):64–72, juin 1992.

[CNSD93] Carolina CRUZ-NEIRA, Daniel J. SANDIN et Thomas A. DEFANTI : Surround-screen projection-based virtual reality : the design and implementation of the cave. *In Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 135–142, New York, NY, USA, 1993. ACM.

[CPC99] G.S. CARSON, R.F. PUK et R. CAREY : Developing the vrml 97 international standard. *Computer Graphics and Applications, IEEE*, 19(2):52–58, 1999.

[CW12] Wojciech CELLARY et Krzysztof WALCZAK : Interactive 3d content standards. *In* Wojciech CELLARY et Krzysztof WALCZAK, éditeurs : *Interactive 3D Multimedia Content*, pages 13–35. Springer London, 2012.

[DB00] W.H. DE BOER : Fast terrain rendering using geometrical mipmapping. *Unpublished paper, available at http ://www. flipcode. com/articles/article geomipmaps. pdf*, 2000.

[DH02] J. DÖLLNER et K. HINRICHS : A generic rendering system. *IEEE Transactions on Visualization and Computer Graphics*, page 99118, 2002.

[DR03] Raimund DACHSELT et Enrico RUKZIO : Behavior3d : an xml-based framework for 3d graphics behavior. *In Proceedings of the eighth international conference on 3D Web technology*, Web3D '03, pages 101–ff, New York, NY, USA, 2003. ACM.

[DWS+97] M. DUCHAINEAU, M. WOLINSKY, D. E SIGETI, M. C MILLER, C. ALDRICH et M. B MINEEV-WEINSTEIN : *ROAMing terrain : real-time optimally adapting meshes*, page 88. 1997.

[Eck98] George ECKEL : *Cosmo 3D™ Programmer's Guide*, volume Document Number 007-3445-002. Silicon Graphics, Inc, 1998.

[FL05a] JYH FUH et WD LI : Advances in collaborative cad : the-state-of-the art. *Computer-Aided Design*, 37(5):571–581, 2005.

[FL05b] J.Y.H. FUH et W.D. LI : Advances in collaborative cad : the-state-of-the art. *Computer-Aided Design*, 37(5):571–581, avr 2005.

[FMB+06] P. FUCHS, G. MOREAU, A. BERTHOZ, J.L. VERCHER, D. AMARANTINI, F. MULTON, G. RAO et F. AUBERT : Le traité de la réalité virtuelle(volume i, l'homme et l'environnement virtuel). *Sciences mathématiques et informatique*, 2006.

[fora] 3d file formats. http ://www.ibrtses.com/opengl/fileformats3d.html.

[forb] 3ds max file formats. http ://wiki.cgsociety.org/index.php/3ds_Max_File_Formats.

[forc] Dxf reference. http ://images.autodesk.com/adsk/files/acad_dxf0.pdf.

[ford] Fbx assets file format. http ://images.autodesk.com/adsk/files/fbxassets.pdf.

[fore] Ogre 3d. http ://www.ogre3d.org/.

[forf] Wavefront obj : Summary. http ://www.british-legends.com/.

[for06] Extensible 3d (x3d) specification. ISO/IEC 19776-1, 2006.

[Gel08] Jean H.A. GELISSEN : ISO/IEC JTC 1/SC 29/WG 11/N9901, mai 2008.

[Gel09] J. H.A GELISSEN : Introduction to mpeg-v. *Journal of Virtual Worlds Research*, 2(3), 2009.

[GHJV95] GAMMA E., HELM R., JOHNSON R. et VLISSIDES J. : *Design patterns : Elements of reusable Object-Oriented Software*. Addison-Wesley, 1995.

[GL09] J. GREGORY et J. LANDER : *Game Engine Architecture*. AK Peters Ltd, 2009.

[Gou71a] H. GOURAUD : Computer display of curved surfaces. Rapport technique, DTIC Document, 1971.

[Gou71b] H. GOURAUD : Continuous shading of curved surfaces. *IEEE transactions on computers*, 20(6):623628, 1971.

[GSV+03] M. GARAU, M. SLATER, V. VINAYAGAMOORTHY, A. BROGNI, A. STEED et M.A. SASSE : The impact of avatar realism and eye gaze control on perceived quality of communication in a shared immersive virtual environment. *In Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 529–536. ACM, 2003.

[Hei62] M.L. HEILIG : Sensorama simulator, 1962. US Patent 3,050,870.

[Hin00] J.D.K. HINRICHS : A generalized scene graph. *Vision, modeling, and visualization 2000 : proceedings : November 22–24, 2000, Saarbrücken, Germany*, 247, 2000.

[HJ08] Shun-Yun HU et Jehn-Ruey JIANG : *Plug : Virtual Worlds for Millions of People*, pages 787–792. IEEE Computer Society, 2008.

[HK10] B. HASLHOFER et W. KLAS : A survey of techniques for achieving metadata interoperability. *ACM Computing Surveys (CSUR)*, 42(2):7, 2010.

[HLT08] A. HENDAOUI, M. LIMAYEM et C. W. THOMPSON : 3d social virtual worlds : research issues and challenges. *IEEE Internet Computing*, 12(1):88,92, 2008.

[Hop96] Hugues HOPPE : *Progressive meshes*, pages 99–108. ACM, 1996.

[JP10] B. JOVANOVA et M. PREDA : Avatars interoperability in virtual worlds. *In Multimedia Signal Processing (MMSP), 2010 IEEE International Workshop on*, pages 263 –268, oct. 2010.

[Jun06] G. JUNKER : *Pro OGRE 3D programming*. Apress, 2006.

[KCK+08] S. KUMAR, J. CHHUGANI, C. KIM, D. KIM, A. NGUYEN, P. DUBEY, C. BIENIA et Y. KIM : Second life and the new generation of virtual worlds. *Computer*, 41(9):46–53, 2008.

[KCS11] N. KATZ, T. COOK et R. SMART : Extending web browsers with a unity 3d-based virtual worlds viewer. *IEEE Internet Computing*, 15(5):15–21, 2011.

[Khr11] KHRONOS GROUP : WebGL. 2011. http ://www.khronos.org/webgl/.

[Koe92] John KOEGEL : On the design of multimedia interchange formats. *In Proceedings of the 3rd International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 12–13, 1992.

[KWD99] F. KUHL, R. WEATHERLY et J. DAHMANN : *Creating computer simulation systems : an introduction to the high level architecture*. Prentice Hall PTR Upper Saddle River, NJ, USA, 1999.

[LE97] David LUEBKE et Carl ERIKSON : View-dependent simplification of arbitrary polygonal environments. *In Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 199–208, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[LG95] D. LUEBKE et C. GEORGES : Portals and mirrors : Simple, fast evaluation of potentially visible sets. *In Proceedings of the 1995 symposium on Interactive 3D graphics*, page 105. ACM, 1995.

[LH04] Frank LOSASSO et Hugues HOPPE : *Geometry clipmaps : terrain rendering using nested regular grids*, pages 769–776. ACM, 2004.

[Liv11] Daniel LIVINGSTONE : Second life is dead, long live second life ? *EDUCAUSE Review*, 46(2), avr 2011.

[LMM96] R. LEA, K. MATSUDA et K. MIYASHITA : *Java for 3D and VRML Worlds*. New Riders, 1996.

[Lop11] Cristina LOPES : Hypergrid : Architecture and protocol for virtual world interoperability. *IEEE Internet Computing*, 15(5):22–29, sept 2011.

[LVG96] Hubert LE VAN GONG : *Paradigmes pour l'interopérabilité entre environnements virtuels = Paradigms for interoperability between virtual environments*. Thèse de doctorat, Université de Paris 06, 1996.

[MB08] K. MCHENRY et P. BAJCSY : An overview of 3d data content, file formats and viewers. Technical Report Technical Report ISDA08-002, National Center for Supercomputing Applications, 1205 W Clark, Urbana, IL 61801, 2008.

[Mea82] D. MEAGHER : Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129–147, 1982.

[Mic11] Laurent MICHAUD : World video game market. Rapport technique, IDATE, jun 2011.

[MOM+11] K. MCHENRY, M. ONDREJCEK, L. MARINI, R. KOOPER et P. BAJCSY : Towards a universal viewer for digital content. *Procedia Computer Science*, 4:732–739, 2011.

[MT95] DC MILLER et JA THORPE : SIMNET : the advent of simulator networking. *Proceedings of the IEEE*, 83(8):1114–1123, 1995.

[MZWG07] Pascal MÜLLER, Gang ZENG, Peter WONKA et Luc J. Van GOOL : Image-based procedural modeling of facades. *ACM Trans. Graph.*, 26(3):85, 2007.

[OJ10] Sixto ORTIZ JR. : Is 3D Finally Ready for the Web ? *Computer*, 43(1):14–16, 2010.

[PBSB08] Nicholas F. POLYS, Don BRUTZMAN, Anthony STEED et Johannes BEHR : Future standards for immersive VR : report on the IEEE virtual reality 2007 workshop. *IEEE Comput. Graph. Appl.*, 28(2):94–99, 2008.

[Pes94] Mark PESCE : Mark pesce, tony parisi – vrml / vrml 97 / x3d 10th anniversary – 3d test panorama of web 3d technologies – real time 3d and interactive media. 1994.

[PF06] Fabio PITTARELLO et Alessandro De FAVERI : *Semantic description of 3D environments : a proposal based on web standards*, pages 85–95. ACM, 2006.

[Pho75] B. T PHONG : Illumination for computer generated pictures. *Communications of the ACM*, 18(6):317, 1975.

[PLH88] P. PRUSINKIEWICZ, A. LINDENMAYER et J. HANAN : Development models of herbaceous plants for computer imagery purposes. *ACM SIGGRAPH Computer Graphics*, 22(4):141–150, 1988.

[PZVBG00] H. PFISTER, M. ZWICKER, J. VAN BAAR et M. GROSS : Surfels : Surface elements as rendering primitives. *In Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 335–342. ACM Press/Addison-Wesley Publishing Co., 2000.

[Rey87] Craig W. REYNOLDS : Flocks, herds and schools : A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21(4):25–34, août 1987.

[RGSS09] Dmitri RUBINSTEIN, Iliyan GEORGIEV, Benjamin SCHUG et Philipp SLUSALLEK : *RTSG : ray tracing for X3D via a flexible rendering framework*, page 4350. Web3D 09. ACM, 2009.

[RHS+98] S. ROETTGER, W. HEIDRICH, P. SLUSALLEK, H. P SEIDEL et G. DATENVERARBEITUNG : *Real-time generation of continuous levels of detail for height fields*, pages 315–322. 1998.

[RL00] S. RUSINKIEWICZ et M. LEVOY : QSplat : A multiresolution point rendering system for large meshes. *In Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 343–352. ACM Press/Addison-Wesley Publishing Co., 2000.

[RLSS10] Michael REPPLINGER, Alexander LÖFFLER, Benjamin SCHUG et Philipp SLUSALLEK : *SORA : a Service-Oriented Rendering Architecture*. 2010.

[Ros89] R.J. ROST : Off-a 3d object file format. *Digital Equipment Corporation Technical Report*, 1989.

[SA97] M. SOTO et S. ALLONGUE : A semantic approach of virtual worlds interoperability. *In Enabling Technologies : Infrastructure for Collaborative Enterprises, 1997., Proceedings Sixth IEEE workshops on*, pages 173–178, 1997.

[SA02] Michel SOTO et Sébastien ALLONGUE : Modeling methods for reusable and interoperable virtual entities in multimedia virtual worlds. *Multimedia Tools and Applications*, 16(1):161–177, 2002.

[SC92] P. S STRAUSS et R. CAREY : An object-oriented 3d graphics toolkit. *In Computer Graphics (Proceedings of SIGGRAPH 92)*, volume 26, pages 341–349. ACM, ACM, 1992.

[Sce]

[SF09] M. SPAGNUOLO et B. FALCIDIENO : 3d media and the semantic web. *Intelligent Systems, IEEE*, 24(2):90–96, 2009.

[SKR+10] K. SONS, F. KLEIN, D. RUBINSTEIN, S. BYELOZYOROV et P. SLUSALLEK : Xml3d : interactive 3d graphics for the web. *In Proceedings of the 15th International Conference on Web 3D Technology*, pages 175–184. ACM, 2010.

[SRH05] Frank STEINICKE, Timo ROPINSKI et Klaus HINRICHS : *A generic virtual reality software systems architecture and application*, page 220227. ICAT 05. ACM, 2005. ACM ID : 1152440.

[SRS08] Juan M. SILVA, Abu Saleh Md. Mahfujur RAHMAN et Abdulmotaleb El SADDIK : *Web 3.0 : a vision for bridging the gap between real and virtual*, pages 9–14. ACM, 2008.

[Sta07] STANDARD ECMA-363 : Universal3D. 2007. http ://www.ecma-international.org/publications/standards/Ecma-363.htm.

[Ste00] Neal STEPHENSON : *Le samourai virtuel*. Librairie générale française, 2000.

[SWGF09] Robert STONE, David WHITE, Robert GUEST et Benjamin FRANCIS : The virtual scylla : an exploration of "serious games", artificial life and simulation complexity. *Virtual Reality*, 13(1):13–25, Mar 2009.

[THS+01] Russell M. TAYLOR, II, Thomas C. HUDSON, Adam SEEGER, Hans WEBER, Jeffrey JULIANO et Aron T. HELSER : Vrpn : a device-independent, network-transparent vr peripheral system. *In Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '01, pages 55–61, New York, NY, USA, 2001. ACM.

[TS91] S.J. TELLER et C.H. SÉQUIN : Visibility preprocessing for interactive walkthroughs. *In Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, page 70. ACM, 1991.

[WNA90] J. WELLING, C. NUUJA et P. ANDREWS : P3d : A lisp-based format for representing general 3d models. *In Proceedings of the 1990 ACM/IEEE conference on Supercomputing*, pages 766–774. IEEE Computer Society Press, 1990.

RÉSUMÉ

Les environnements virtuels 3D sont aujourd'hui utilisés dans de nombreux domaines et couvrent de multiples usages qui vont du divertissement à l'apprentissage en passant par la conception pour l'industrie. Si leur nombre ne cesse de croître chaque année, un problème récurrent freine leur adoption et leur diffusion. Il est en effet aujourd'hui très difficile d'importer des données dans un environnement virtuel autre que celui pour lequel ces données ont été créées. Ceci entraîne de nombreux autres problèmes qui ralentissent la création d'environnements virtuels 3D, augmente leur coût de production et limite leur accès aux utilisateurs. Nous présentons ici une solution qui se propose de réconcilier les modèles existants et à venir, de façon à rendre interopérables les contenus et les composants logiciels des environnements virtuels 3D. Cette solution repose sur deux éléments complémentaires : une architecture logicielle et un modèle de conteneur de formats 3D. Nous avons appelé notre système l'adaptateur de graphes de scène ou SGA. Il s'agit d'une architecture générique et modulaire qui permet le chargement de plusieurs formats 3D dans la plupart des composants logiciels des environnements virtuels 3D. Le SGA a non seulement pour rôle d'adapter tous les graphes de scène encodés dans les fichiers 3D en graphes de scène pour les composants qui sont utilisés par l'environnement virtuel mais également de gérer la synchronisation de ces graphes de scène au cours du déroulement de l'application de l'environnement virtuel. Nous avons réalisé une implémentation de cette architecture ainsi que plusieurs instanciations qui permettent le chargement de fichiers au format X3D et COLLADA dans une application dont le rendu se base sur le moteur de rendu graphique Ogre3D et le moteur physique Bullet. Ceci nous a permis de démontrer la faisabilité de cette solution et d'évaluer ces performances. Notre modèle de conteneur de formats 3D permet non seulement de composer des scènes faites à partir de plusieurs fichiers 3D mais aussi de combiner leurs fonctionnalités et de les faire interagir dans l'environnement virtuel. Ce modèle est appelé 3DFC pour conteneur de fichiers 3D et il repose sur le système du SGA pour l'adaptation des graphes de scène encapsulés dans les fichiers référencés par le conteneur. Nous avons réalisé une instanciation pour le modèle 3DFC que nous avons intégrée à notre implémentation du SGA. Nous avons ainsi pu mixer dans une même scène des modèles X3D et COLLADA et combiner les fonctionnalités d'interactions offertes par X3D avec les propriétés physiques autorisées dans COLLADA.

ABSTRACT

3D virtual environments are used in many domains with application ranging from entertainment to learning through computer-aided design for industrials. Even if they grow in number each year, a remaining problem impedes their widespread adoption and distribution. It is indeed very difficult to import data into a virtual environment other than the one it has been designed for. This situation leads to several other problems that hinder the creation of new virtual environments, increase their production cost and restrict user access.

Here, we present a solution based on reconciliation of existing and coming models in order to make 3D virtual environments contents and software components interoperables. This solution relies on two complementary elements : a software architecture and a 3D file container model.

We called our system the SGA for Scene Graph Adapter. It is a generic and modular architecture which allows the loading of several 3D formats in most 3D virtual environments software components. The SGA not only adapts scene graphs encoded in 3D files into scene graphs used by rendering software components but also manages the synchronization between the involved scene graphs at runtime of a 3D virtual environment. We have made an implementation of this architecture as well as several instantiations that allows the loading of X3D and COLLADA files into an application which relies on the Ogre3D rendering engine and the Bullet physics engine. It allows us to demonstrate the feasibility of our solution and also to assess its performances.

Our 3D file container makes it possible not only to compose 3D scenes made of several 3D files but also to combine their features as well as make them interact in the rendered environment. This model has been called 3DFC for 3D File Container and it relies on the SGA for the adaptation of the encapsulated scene graphs from the referenced files. We have made an instantiation for 3DFC that has been integrated into our SGA implementation. Thus, we are able to mix X3D and COLLADA models in a single scene and to combine interaction features provided by X3D with physics properties offered by COLLADA.

INSA RENNES

Institut National des Sciences Appliquées de Rennes

20, Avenue des Buttes de Coësmes . CS 14315 . F-35043 Rennes Cedex

Tel : 02 23 23 82 00 – Fax : 02 23 23 83 96